2004

DEVELOPMENT AND EVALUATION OF A CONTROLLER AREA NETWORK BASED AUTONOMOUS VEHICLE

Matthew John Darr
University of Kentucky, mdarr@bae.uky.edu

ABSTRACT OF THESIS

DEVELOPMENT AND EVALUATION OF A CONTROLLER AREA NETWORK BASED AUTONOMOUS VEHICLE

Through the work of researchers and the development of commercially available products, automated guidance has become a viable option for agricultural producers. Some of the limitations of commercially available technologies are that they only automate one function of the agricultural vehicle and that the systems are proprietary to a single machine model.

The objective of this project was to evaluate a controller area network (CAN bus) as the basis of an automated guidance system. The prototype system utilized several microcontroller-driven nodes to act as control points along a system wide CAN bus. Messages were transferred to the steering, transmission, and hitch control nodes from a task computer. The task computer utilized global positioning system data to determine the appropriate control commands.

Infield testing demonstrated that each of the control nodes could be controlled simultaneously over the CAN bus. Results showed that the task computer adequately applied a feedback control model to the system and achieved guidance accuracy levels well within the range sought. Testing also demonstrated the system's ability to complete normal field operations such as headland turning and implement control.

KEYWORDS: Controller Area Network, Distributed Control Systems, Autonomous Vehicle, Precision Agriculture, Embedded Microcontroller Systems

Matthew John Darr

March 5, 2004

DEVELOPMENT AND EVALUATION OF A CONTROLLER AREA NETWORK BASED AUTONOMOUS VEHICLE

By

Matthew John Darr

Dr. Timothy S. Stombaugh
Director of Thesis

Dr. Dwayne R. Edwards
Director of Graduate Studies

March 5, 2004

THESIS

Matthew John Darr

The Graduate School

University of Kentucky

2004

DEVELOPMENT AND EVALUATION OF A CONTROLLER AREA NETWORK BASED AUTONOMOUS VEHICLE

THESIS

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in
Biosystems and Agricultural Engineering
in the College of Engineering
at the University of Kentucky

By

Matthew John Darr

Lexington, Kentucky

Director: Dr. Timothy S. Stombaugh, Assistant Extension Professor of
Biosystems and Agricultural Engineering
Lexington, KY

2004

ACKNOWLEDGMENTS

First and foremost, I would like to thank God for granting me the ability to successfully complete this project.

I would like to thank my wife Kristi for all of her love and support throughout this project. I would not have been successful without her help and encouragement. She was understanding and supportive as I spent long evenings at the office and did whatever she could to make our time in Lexington wonderful. Thank you!

I also wish to acknowledge the support and understanding of my extended family over the past two years. To my parents, grandparents, brother, and close friends, thank you all for your calls, visits, and encouragement!

Dr. Tim Stombaugh served as my major advisor and was instrumental in assisting with technical issues and theories as well as acting as a sounding board for new ideas. Throughout this project he taught me how to conduct and analyze engineering research. He also taught me what it means to be a professional in the field of engineering and I will take this experience along with me throughout my future career.

Dr. Scott Shearer served on my committee and often provided technical assistance. He helped me understand what it means to be a part of the academic society and what our role is in shaping the world of agricultural.

Dr. Jim Lumpp also served on my committee and taught me the basic principles necessary to successfully complete this work. He also provided valuable insight when my technical abilities were still very raw.

Dr. Michael Lichtensteiger was my undergraduate advisor while at The Ohio State University. He was instrumental in teaching me the basic fundamentals of engineering as well as advising me during my first introduction into engineering experimentation. It was his advice that directed me into a master's program and into the field of academia.

Finally, I would like to thank the rest of the faculty, staff, and graduate students at the University of Kentucky. I truly enjoyed my time in Lexington and will value my experiences here for years to come.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF FILES

Darr_MST.pdf……1.11 MB

Chapter 1: Introduction

Over the past several years, technology has continued to play an increasing role in agriculture. The industry has recently seen the advent and development of many types of autonomous vehicles ranging from planters to sprayers to harvesters. These vehicles have all sustained different levels of autonomous operation. Some are capable of fully autonomous field operations while others were developed for specific control operations such as autosteering (Reid et al., 2000). While these advances have been impressive, commercially available products have also come at a substantial cost to the farmer. These products range in cost from $20,000 to $50,000 depending on their accuracy and functionality. The largest component of the system price is the level of global positioning system (GPS) accuracy desired.

Many systems developed for fully autonomous control rely on a central processing unit to coordinate all vehicle operations and to carry out all control routines. This design places high demands on the processing unit, which in turn triggers high unit cost.

There has also been a recent increase in the number of electronic components on agricultural equipment. During normal field situations, operators must interact with spray rate controllers, variable rate planter controllers, and implement system controllers as well as controls for normal vehicle operation. Attempts have been made to create a standard communication link within all agricultural equipment but have thus far failed within the US. The most common ideology for a worldwide equipment communication protocol is the International Organization for Standardization (ISO) 11783 standard, which was designed for agricultural and construction equipment. This standard utilizes the Controller Area Network (CAN bus) 2.0B protocol to transmit serial data between networked control systems (Stone et al., 1999b). The CAN bus system was designed to link multiple electronic control units (ECUs) over a single data bus and inherently lends itself to becoming the backbone of a distributed control system for autonomous vehicle operations because of its high data transmission rate, expandability, and reliability.

The overall goal of this project was to design and test an autonomous vehicle based on a CAN distributed control scheme and to evaluate the expandability of a CAN

system. The functionality of the autonomous system will be tested through bench top studies as well as through field operation routines.

Chapter 2: Literature Review

A. Controller Area Network (CAN Bus)

Robert Bosch GmbH designed the Controller Area Network in 1986 upon request by Mercedes to develop a system that would allow for communication between three electronic control units (ECU). It was noted that a standard UART communication could not complete the task because it only allowed for point-to-point communication. Although the CAN bus was originally designed for automotive applications, it has been applied to many areas of automation and control. CAN has been used in applications including warehouse shipping automation, packaging machines, medical devices including X-ray collimators and patient tables, and building controls including alarm and sprinkler systems. The unique aspect of a CAN network is that each message is preceded with an identifier that is unique to the transmitting controller and that multiple controllers can communicate over a single two-wire bus. Two wires are required for the node to assert the two different voltage levels defined by the CAN protocol. If two messages are sent simultaneously, an automatic arbitration process ensures that the highest priority message is sent first. The lower priority message then has the opportunity to retransmit upon completion of the first message. Incoming messages are filtered by the ECUs based on the unique message identifier of the sender (CAN-CIA).

There are three separate CAN standards: CAN Version 1.0, Version 2.0A (Standard CAN), and Version 2.0B (Extended CAN). The main difference in the three standards is the length of the identifiers that precede each message. All work presented in this manuscript is based on the CAN 2.0B standard.

B. ISO 11783

i. Introduction

The ISO 11783 document was created to standardize electronic communication on agricultural tractors and equipment. The ISO 11783 standard is comprised of thirteen documents (Table 1). CAN 2.0B was chosen for the communication protocol

because of its growing use in mobile equipment. The ISO 11783 standard extends the definition of the CAN 2.0B protocol and specifies many parameters concerning the serial communication and hardware connections. Much of ISO 11783 follows the exact specifications of the Society of Automotive Engineers (SAE) standard J1939 in an attempt to make the two standards compatible. SAE J1939 is an automotive network communication standard that also uses CAN 2.0B. The SAE J1939 standard clearly defines the physical parameters and message types used in automotive communication systems. These standard messages include vehicle speed, engine temperature, and fuel consumption, among many others (SAE J1939). Likewise, DIN 9684 is a European standard for agricultural equipment networks that uses the CAN 1.0 protocol (DIN 9684). Both SAE J1939 and DIN 9684 were used as models for ISO 11783 development (Stone et al., 1999b). The first stage of ISO 11783 standardization was to define a physical layer. This specified the type and layout of the physical wire used for communication.

Table 1. ISO 11783 Documents

Document	Description	Form
ISO 11783 Part 1	General Standard	Draft
ISO 11783 Part 2	Physical Layer	Completed
ISO 11783 Part 3	Data Link Layer	Completed
ISO 11783 Part 4	Network Layer	Completed
ISO 11783 Part 5	Network Management	Completed
ISO 11783 Part 6	Virtual Terminal	Draft
ISO 11783 Part 7	Implement Message Layer	Completed
ISO 11783 Part 8	Drive Train Message Layer	Draft
ISO 11783 Part 9	Tractor ECU Interconnection Unit	Completed
ISO 11783 Part 10	Task Controller Applications Layer	Draft
ISO 11783 Part 11	Mobile Agriculture Data Element Dictionary	Draft
ISO 11783 Part 12	Diagnostics	Draft
ISO 11783 Part 13	File Server	Draft

ii. Physical Layer

The physical defined by the ISO 11783 standard is unshielded twisted quad cable. Two conductor lines, CAN_H and CAN_L, are used as the communication lines.

The other two lines, TBC_PWR and TBC_RTN, are used to provide power to the terminating biased circuits. These terminating biased circuits provide an equivalent resistance of 120 ohms at each end of the CAN bus and suppress any electrical signal reflections (ISO 11783/2).

The CAN bus is unique among agricultural communication protocols because the voltage levels referring to logic one and logic zero are measured between two signals of positive voltage, rather than one positive signal and a common ground. During a recessive bus time (logic one), both CAN_H and CAN_L are 2.5 volts. During a dominant bus time (logic zero), CAN_H is 3.5 volts and CAN_L is 1.5 volts (Figure 1). This creates a dominant differential voltage of 2 volts (Equation 1).

$$V_{diff} = CAN\_H - CAN\_L \qquad (1)$$

Where: V_{diff} = Differential Voltage Level

CAN_H = CAN High Voltage Level

CAN_L = CAN Low Voltage Level

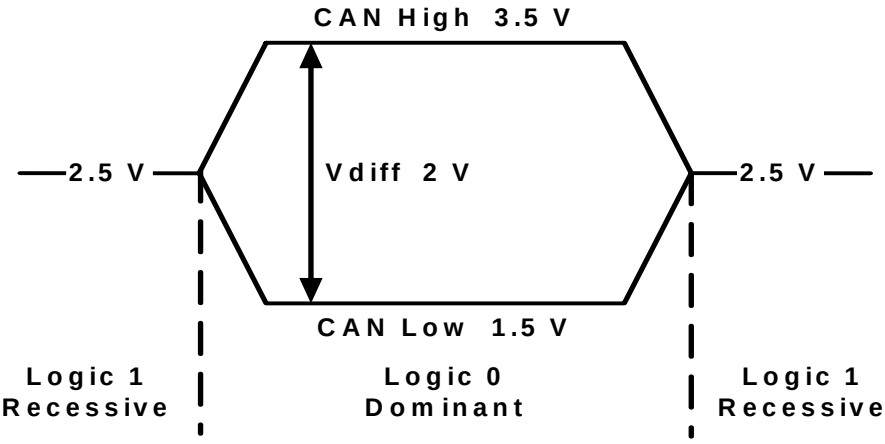


Figure 1. Schematic of CAN Logic Levels

ISO 11783 places strict guidelines on bus length and configuration (Figure 2) . The overall length of any sub-bus (L) cannot exceed 40 meters. Node stub attachments along the bus (S) cannot exceed 1 meter. The distance between each node stub along the bus (d) must exceed 0.1 meters and total node attachments per bus must be less than 30 units (Table 2). All buses are run in a straight line or S shape configuration; thus, no T buses are permitted (Figure 2). This physical limitation reduces the risk of message transmission error due to cable reflected waves (ISO 11783/2).

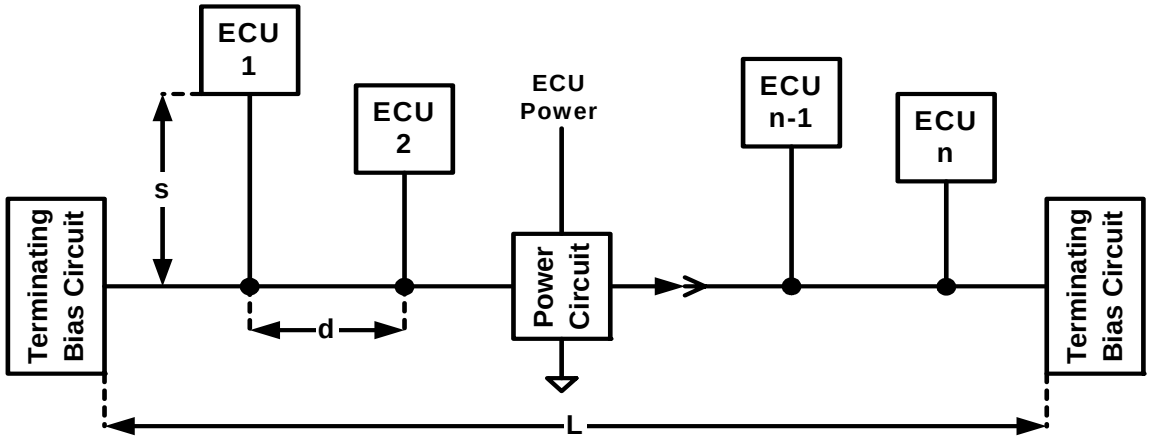


Figure 2: ISO 11783 Physical Bus Layout

Table 2: ISO 11783 Physical Bus Parameters

Parameter	Symbol	Min	Nom	Max	Unit
Bus Length	L	0		40	m
Stub Length	S	0		1	m
Node Distance	D	0.1		40	m

The virtual terminal is a multi-purpose task computer and application controller that is designed for CAN-based operations. ISO 11783 calls for standardized bus connectors in the vehicle cab to service this virtual terminal. A standard bus breakaway and terminating bias connector must also be present wherever implements and prime movers are linked. This connector allows communication from the prime mover to the implement when connected, and also provides bus termination bias when disconnected. Furthermore, a standard diagnostics connector must be present to provide error troubleshooting and analysis (ISO 11783/2).

Twenty four failure mode cases are discussed in the ISO document to indicate that communication can continue during limited instances of bus failure (ISO 11783/2). These failure cases include shorting one transmission line to ground and interrupting one transmission line.

iii. ISO 11783 Data Link Layer

The ISO 11783 Data Frame is defined to be up to 128 bits and is broken into an identifier and data section (Figure 3). The identifier is a series of 29 bits that designates the priority and address (identity) of the message. In the CAN protocol, a zero bit has a higher priority than a one bit, so smaller numeric identifiers have higher message priority (ISO 11783/3).

If two or more nodes begin transmission simultaneously, a bitwise arbitration procedure is executed on the identifier frame of the message. If a transmitter detects a dominant bit on the bus after a recessive bit was transmitted, the node losses arbitration and reverts to being a receiver. If multiple nodes assert a dominant bit simultaneously then the arbitration process continues until only one node remains transmitting on the bus (Figure 4). CAN 2.0B utilizes a 29 bit identifier, thus there are $2^{29} - 1$ possible node definitions (Bosch, 1991).

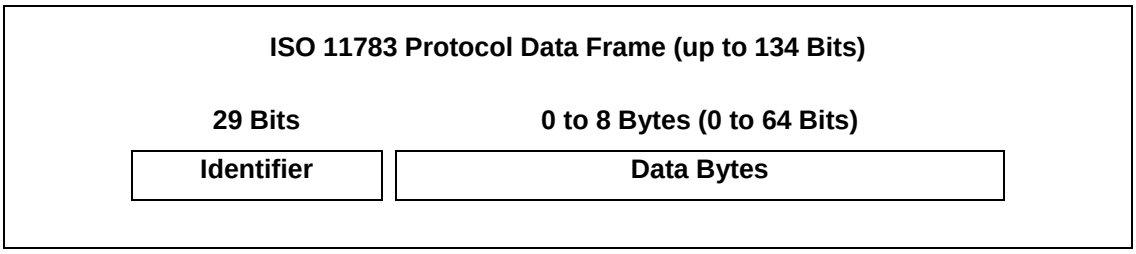


Figure 3: ISO 11783 Data Frame

	Transmitted Bits							
Node 1	0	0	1	0	1	Listen Only		
Node 2	0	1	Listen Only					
Node 3	0	0	1	0	0	1	1	0

Figure 4. Multi-node Arbitration Process

Priority			Reserved	Data Page	PDU Format	PDU Specific Destination Address	Source Address
P 1	P 1	P 1	R 1	DP 1	PF 8	PS(DA) 8	SA 8

Figure 5: ISO 11783 Identifier String

The ISO 11783 identifier string is broken into six segments: priority, reserved, data page, PDU format, PDU specific (destination address, group extension, or proprietary), and source address (Figure 5). Each segment serves its own role while also merging to form the identifier for the node (ISO 11783/3).

The priority section contains three bits, which are used to signify the priority of the message. Again, a lower number holds a higher priority on the CAN bus. Message priority levels (Table 3) are predefined to ensure critical messages will incur limited latency times (ISO 11783/3).

Table 3. Message Priority Assignments (Reprinted from ISO 11783-3 5.2.1)

Priority Field Number	Priority Level	Default System
000	0 - Highest	
001	1	
010	2	
011	3	Control Systems
100	4	
101	5	
110	6	Proprietary Information
111	7 – Lowest	

The reserved data section is one bit and is reserved for future use by ISO. The node controller designates this bit as zero upon transmission of a new message (ISO 11783/3).

The data page section contains one bit of information, which acts as an auxiliary bit to the parameter group number. Page one parameter group numbers are designated by setting the data page bit to one. All parameter group numbers in page zero (data page bit = 0) must be used before page one is activated (ISO 11783/3).

The protocol data unit (PDU) format section (Table 4) defines whether the PDU specific section is a destination address or a group extension. Three groups were

created: PDU1 format, PDU2 format and global format. PDU1 format allows for both node specific and global message destinations. The PDU2 format allows only global message destinations. Global format is a message that all nodes must acknowledge and read (ISO 11783/3).

Table 4. PDU Format Description (Reprinted from ISO 11783-3 5.2.5)

PDU Format	PDU Format Field Value	Destination
PDU1	0 to 239	specific or global
PDU2	240 to 254	global
Global	255	global

The PDU format section defines the PDU specific section. A PDU1 format signifies that the PDU specific section contains the destination address. This allows the message sender to specify a particular recipient. If the destination address is 255 then all nodes must listen and respond to the message. A PDU2 format signifies that the PDU specific section contains the group extension field. The group extension is a global designator that targets multiple nodes. This message type is used to send data or commands to several nodes simultaneously (ISO 11783/3).

The source address field is a specific address for each node on the CAN bus. Only one controller on a network can hold a specific source address. As an eight-bit field, the maximum amount of nodes per bus is 256. In application this number is limited to 254, because the null (0) and global addresses (255) are not used (ISO 11783/3). While the Data Link Layer allows for 254 unique nodes, the physical layer limits bus implementations to a maximum of 30 nodes.

The parameter group number (PGN) is defined as the reserved data bit, data page bit, PDU format bits, and destination address bits (ISO 11783/3). ISO 11783 defines many PGNs for standard operating nodes such as engine, transmission, axles, brakes, and implement lighting. Several of these functions also have predefined source addresses. Those that do not have predefined source addresses must dynamically register themselves on the bus network (ISO 11783/3).

Bosch GmbH designed the CAN to transmit at bus speeds of up to 1 Mbits/sec. ISO 11783 limits this speed on agricultural buses to 250 kbits/sec in an attempt to maintain harmony with SAE J1939. This baud rate corresponds to a bit time of 4 μs. ISO 11783 also defins bus synchronization to occur on the rising signal edge, thus during a transition from a recessive to dominant bit. A single bit sampling method must be used and should cover 80 ± 3% of the total bit time as referenced from the start of the bit (ISO 11783/2).

iv. Network Management

Network management must be addressed when linking multiple bus networks within one tractor-implement unit. A bridge node must be present at each location where two or more buses converge. The bridge is responsible for managing traffic on each bus. A bridge retransmits only the information that is vital to operations on other bus networks. This limits the amount of traffic on each bus and decreases the overall latency within the system (ISO 11783/5).

v. Application in Precision Agriculture

Several manufacturers have implemented partial networks or CAN-based systems on their equipment in the past several years. John Deere (Moline, IL) has implemented ECU-based control networks since the introduction of the 7000 series tractor in 1992 (Stone et al., 1999b). The Genesis series tractor from New Holland (New Holland, PA) reported the use of a CAN-based network in 1994 (Young, 1993). Ag-Chem Equipment Company Inc. (Duluth, GA) patented the use of network-based control systems for multi-product application, which has been used in their Falcon control systems (Monson and Dahlen, 1995).

vi. Simulation and Testing

Simulations have shown that normal agricultural machinery configured with an ISO 11783 based CAN bus will produce average message latency of less than 6 msec with proper prioritization. When bus loads increased to 80% capacity, messages with low priorities could see latency times approaching 70 ms (Hofstee and Goense, 1999).

Hofstee and Goense (1999) also showed that average latency associated with passing through a network bridge during normal bus load was between 2.5 and 2.6 msec with a maximum latency of 3.6 msec for messages with a high priority.

C. Autonomous Off-road Vehicles Control and Autosteering

Researchers and agriculturalists have pursued automatic vehicle control for many years. In fact, patents dating back to 1924 detail methods of automatic guidance in furrow strips (Willrodt, 1924). With the advent of precision agriculture and GPS capabilities, vehicle-based guidance again became a focus of researchers. Researchers at the University of Illinois showed that 16 cm steady state error straight-line accuracy could be achieved while traveling up to 6.8 m/sec (Stombaugh et al., 1999). Benson et al. (1998) showed that at slower speeds, a geomagnetic direction sensor (GDS) could reduce the straight-line steady state error to 1 cm.

Machine vision has also been incorporated into control systems. Work by Carnegie-Mellon University and NASA demonstrated that vision-based systems could be used for autonomous control (Ollis and Stentz, 1996). A camera mounted on a hay-windrowing machine was used to identify the cut/uncut edge within the hay crop and guide the windrower to stay along this path. Benson et al. (2001) used a single cab mounted camera along with grain head mounted cameras to develop steering control systems for combine harvesters.

Row crop vision systems have been developed to identify individual plants and determine a path to stay within the plant rows. Near infrared sensors have been used to classify the soil and row crops separately and provide a guidance directrix to a control algorithm (Reid and Searcy, 1987).

Many kinematic vehicle models have been developed to determine the path a vehicle will follow based on a specified steering angle (Ge, 1987). These kinematic models are often referred to as bicycle models, because they lump a four wheel vehicle into a two wheeled model. These models are applicable if four conditions are met: 1) level operating surfaces, 2) constant steering rate, 3) $\pm 20^{\circ}$ maximum steering angle, and 4) a rigidly fixed rear mounted implement (Choi et al., 1990). Furthermore, side forces and slip angles could be ignored when the vehicle was traveling less than 12

km/h (Wong, 1978; Owen, 1982). Julian (1971) found that lateral fluctuations of the front end of the vehicle less than ±5 centimeters would create negligible deviations of a rear-mounted implement. All kinematic and bicycle model references in this text are based on the work of Grovum and Zoerb (1970), who derived a mathematical model describing tractor dynamics. This work should be used as a detailed reference to the kinematic model work presented here.

While research has been completed on autonomous vehicle guidance, few studies have focused on end of row turning methods. Noguchi et al. (2001) has shown that a spline function can be used to estimate the turning function of a vehicle at the end of a row. Other work has discussed the importance of full field operations, but has not addressed the problem of vehicle guidance during turning (Han and Zhang, 2001).

D. Distributed Control Systems

Distributed control systems have been used in several sub-network functions of automatic equipment. Distributed control systems have been used to identify individual weeds within row crops and operate appropriate control systems to apply pesticide to the weeds (Tian et al., 1999). Similarly, Stone et al. (1999a) used a CAN-based distributed control system to control the application rate of liquid fertilizer depending on the fertility of the wheat as determined from a near-infrared reflectance sensor.

Wei et al. (2001) proved that a CAN bus could be used as the foundation for a distributed weed control system. Successful testing was documented along with failures associated with error processing and recognition. Microcontroller use in distributed control systems was demonstrated by controlling low-pressure plant growth chambers at Texas A&M University (Brown and Lacey, 2002). Six microcontrollers were successfully integrated into a distributed control system to monitor and control pressure within a vacuum chamber based on a given set point.

Chapter 3: Objectives

The goal of this project was to evaluate the potential of a CAN bus to be used as the communication network for a distributed control system on an autonomous field vehicle. This project also evaluated whether an inexpensive gyroscope and compass

could assist with autonomous vehicle operations. Furthermore, guidance control algorithms were developed based on a kinematic bicycle model and a digital PID controller.

These project goals were accomplished through implementation of the following sub-objectives.

1. Design a modular distributed control system
2. Interface the electronic control units on the CAN bus with sensors and controls specific to the autonomous operation of a prime mover vehicle
3. Develop a guidance algorithm to control an autonomous test vehicle
4. Demonstrate the ability of the vehicle to complete a typical field operation.

Chapter 4: Results & Discussion

A. Design of a Modular Distributed Control System

The requirements for the distributed control system were as follows:

1. A dedicated node was located at each control location
2. All nodes were able to communicate to all other nodes in a multi-master communication system
3. Each node had the capability to implement control routines and interface with feedback sensors
4. The physical layout and enclosure unit was identical for each node
5. There was an RS232 interface to the communication system to allow for task controller interaction.

There were several multi-master based communication systems available to use for this project, including CAN, Inter-Intercomputer Communications (I2C), and Local Area Network (LAN). The CAN 2.0B protocol was chosen from this list as the communication network for the distributed control system. Not only did the CAN system fully satisfy the objective of a multi-master network, but it also enabled simple implementation of dedicated control systems at each node location. CAN systems have also been shown to be very reliable when used in harsh operating environments such as those found on automotive or agricultural vehicles. Many microcontrollers were available with internal CAN engines, which simplified implementation and kept the

overall system cost relatively low. The ISO 11783 standard was followed throughout this project whenever applicable. A bus baud rate of 250 kbits/sec was used to maintain compliance with the ISO 11783 standard.

The microcontrollers required for this project had to execute generic control routines and interface to external sensors to fulfill the distributed control requirements. Specifically, the microcontroller had to have the following capabilities:

1. At least 4 Kbytes of program memory
2. At least 1 Kbyte of RAM
3. At least 3 channels of 10-bit analog to digital conversion
4. Internal CAN engine
5. Hardware pulse width modulation
6. Universal Asynchronous Receive and Transmit (UART) port.

The microcontrollers utilized in the distributed control system were PIC18F258 microcontrollers produced by Microchip (Chandler, AR). This chip provided capabilities that met or exceed the requirements of the project, including an internal CAN module, 5 channels of 10-bit analog to digital conversion, 25 mA sink/source current loads, 8 x 8 single cycle hardware multiplier, 16-bit counter/timer, hardware pulse width modulation, 32,000 bytes of FLASH memory, 1600 bytes of SRAM, and 256 bytes of EEPROM.

The initial requirements dictated that each node must be physically identical, thus each node had to be configured as a generic multi-purpose node. A printed circuit board (Figure 6) was designed to serve as the general board for all ECUs. The board incorporated all external components necessary for microcontroller operation including an oscillator, multiple capacitors for supply voltage regulation, and a reset button. A 20 MHz oscillator was selected to provide timing to the microcontroller on most nodes. This provided single instruction cycle execution time of 0.2 μsec.

A standard input voltage of 12 volts was chosen so that the ECU could be connected directly to a vehicle's battery. The 12 volts was regulated to 5 volts by using an integrated circuit voltage regulator (LM7805), which provided a maximum source current of 1 amp. Two rows of screw terminal connectors were incorporated to provide access to each input and output pin on the microcontroller. Two 5 volt auxiliary power terminals were incorporated into the design to power external sensors (Figure 6).

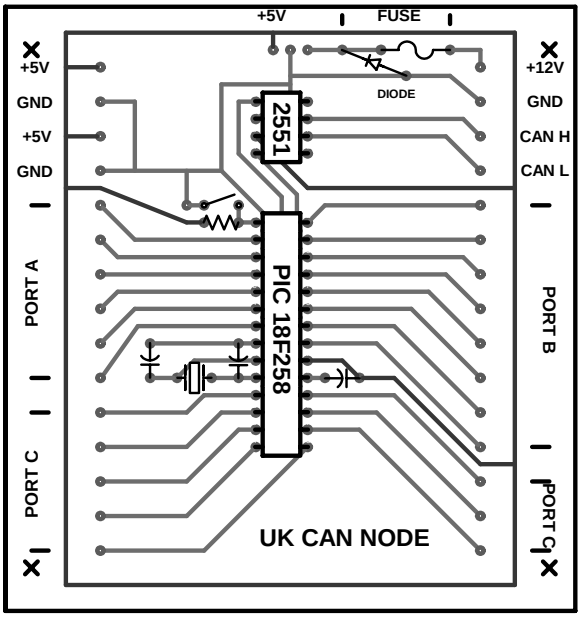


Figure 6. ECU Printed Circuit Board

The microcontroller provided the logic operations for CAN communication, but could not drive the differential voltages required for physical implementation of the communication protocol. An MCP2551 transceiver, also produced by Microchip (Chandler, AR), was incorporated into the node to provide switching between the digital TTL logic of the microcontroller and the differential output required on the CAN bus (Figure 7). The full circuit diagram for the generic CAN node can be seen in Appendix D.

The nominal CAN bus voltage was set based on the operating voltage of the transceiver chip. For this project, the nominal bus voltage was set at 2.5 volts, thus following the ISO 11783 specification.

The transceiver also acted as a buffer for the ECU and prevented transient voltage spikes on the CAN bus from reaching the microcontroller. The slope that the transmission bits rise and fall (slew rate) was also controlled via an external resistor network on the transceiver. The slew rate setting for a particular bus was based on the length and nominal voltages on the bus. Because a short bus length and a low nominal bus voltage were utilized in this project, no slew rate control was implemented.

Limited bus fault protection was incorporated into the transceiver. The transceiver disabled the CAN output lines if an extended low voltage state was sensed on the transmit pin. This prevented the bus from being corrupted with bad data if one of the ECUs malfunctioned. The transceiver reinitiated on the first rising edge of the transmit pin.

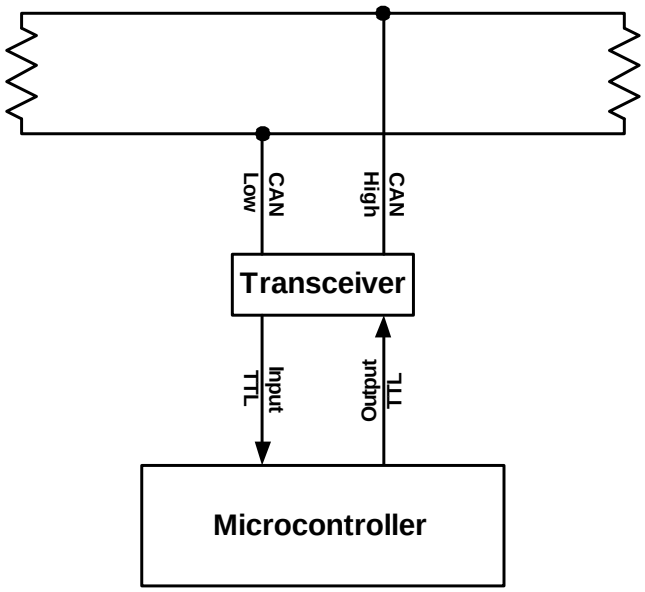


Figure 7. Transceiver Link Between TTL Logic and CAN Logic

A fuse and diode circuit was incorporated before the voltage regulator to protect the board against a reversed polarity supply voltage input. If the supply voltage polarity was reversed, the current would flow through the diode and blow the fuse, thus eliminating any possible damage to the printed circuit board and components. The fuse also provided over-current protection to the circuit board.

The generic printed circuit board fulfilled a portion of the generic node requirement. A black plastic enclosure completed the rest of the requirement and made each node interchangeable. The plastic enclosure was purchased from Futurlec (www.futurlec.com) and measured 5.2” long by 3.5” high by 1.7” deep. Holes were machined in the top and bottom of these boxes to facilitate the physical connection of the printed circuit board to the CAN bus and any external sensors.

The bus cable that connected each ECU consisted of nine wires (Figure 9). Four wires followed the physical layer standard of ISO 11783. Two additional wires provided a 12 volt supply voltage and ground to the ECUs. The three remaining wires were

unused in the initial design, but provided potential for future expansion. The ECU power supply lines were insulated 16 gauge stranded copper wire. All other communication wires were 22 gauge twisted pair stranded copper wire.

Two options were available to link each ECU together. One option consisted of linking each ECU via a single connector attached to the CAN bus. This would reduce the overall connector cost, but would require the precise location of each node be specified to designate the spacing between each connector. A second option was used for this project. Two connectors were incorporated into each ECU to link to the CAN bus. The CAN bus lines were then connected internally in the node to allow for a continuous bus (Figure 8).

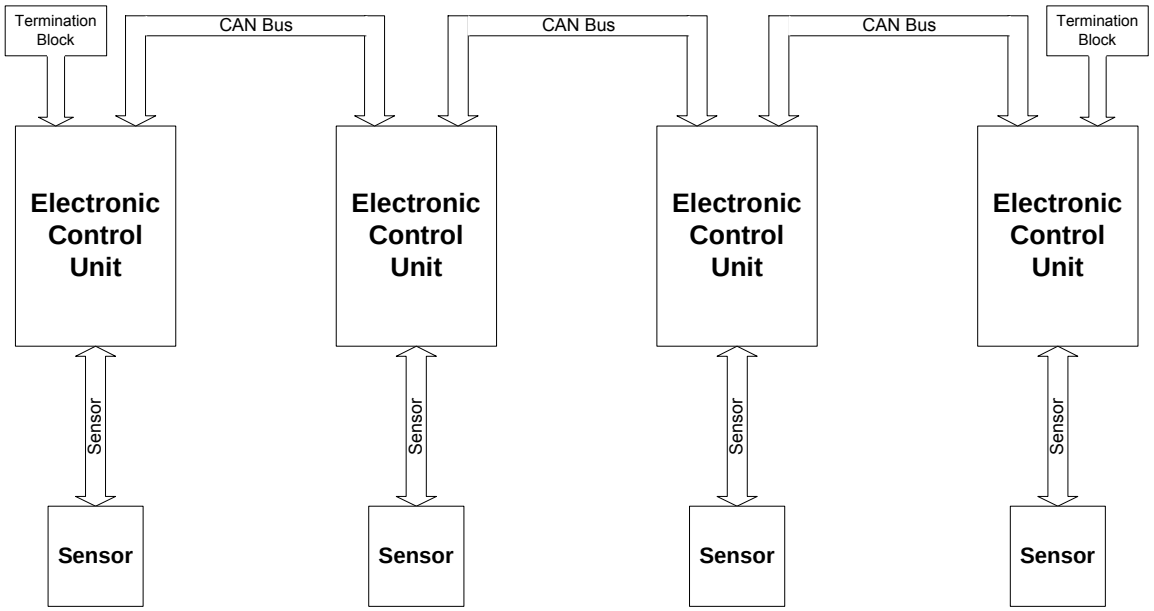


Figure 8. Controller Area Network Layout

Amphenol (Wallingford, CT) 9 pin plastic connectors (Figure 9) with positive lockdown were chosen as the standard connector for this project. The positive lockdown connection resisted loosening that could be caused by mechanical vibrations from the host vehicle. Although not waterproof, the connections were water resistant. A terminating bias connector as defined by ISO 11783 was used as the termination connection at each end of the bus. These connectors were supplied to the project by AGCO Corporation (Duluth, GA).

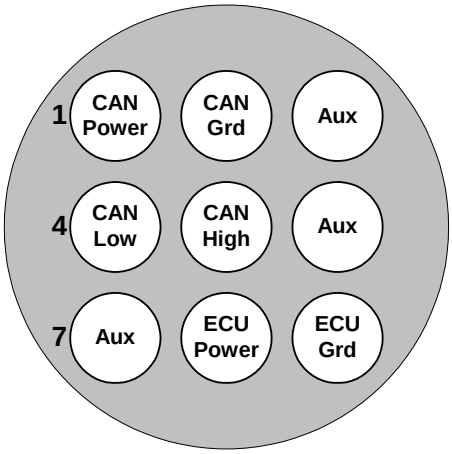


Figure 9. Amphenol Connection Diagram

The final generic node design (Figure 10) included two CAN bus connections on one end of the enclosure. The opposite end of the enclosure housed one or two connections for interfacing with external sensors.

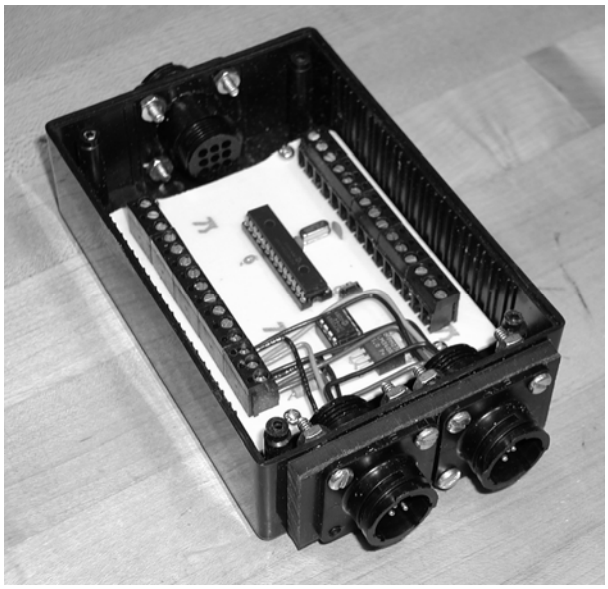


Figure 10. Electronic Control Unit

Machine code for the microcontroller was compiled using the PIC Basic Pro Compiler® and written to the microcontroller using the EPIC® programmer, both products of MicroEngineering Labs, Inc. (Colorado Springs, CO).

A CAN to RS232 bridge was designed to interface the CAN message information with a task computer. The bridge was designed by configuring a specialized high-speed serial communication ECU. The bridge node had a single objective: to receive all messages from the CAN bus at a baud rate of 250 kbits/sec and retransmit the message via an RS232 link to the handheld data collection unit at a baud rate of 256 kbits/sec. By retransmitting the message at a rate faster than the incoming messages, the microcontroller ensured that the CAN receive buffers would not overflow with incoming messages. The 256 kbits/sec baud rate was the fastest allowable serial data rate within the Visual Basic programming environment. While this node utilized the same microcontroller as all other nodes, it contained a specially sized external oscillator (16.384 MHz) to create the serial bit timing of 256 kbits/sec. The CAN messages were retransmitted via RS232 beginning with a leading identifier ($), followed by the source address of the message and the eight message data bytes in a comma-delineated fashion. The message concluded with an ending identifier (#) (Table 5).

Table 5. Message Structure for CAN-RS232 Bridge Operation

Source Address	10
Data Byte 0	134
Data Byte 1	24
Data Byte 2	56
Data Byte 3	64
Data Byte 4	0
Data Byte 5	0
Data Byte 6	0
Data Byte 7	0

$10,134,24,56,64,0,0,0,0#

If one chip had been used to perform both RS232 receiving and transmitting, then hardware serial buffers and interrupt handling would have to have been utilized. The PIC18F258 UART buffers were only 1 byte in size and would have overflowed if a new message was received from the task computer at the same time that a new CAN message was being transmitted to the task computer. A second ECU was developed to

perform bridge operations in the opposite direction. It received messages from the task computer and retransmitted the message over the CAN bus (Figure 11).

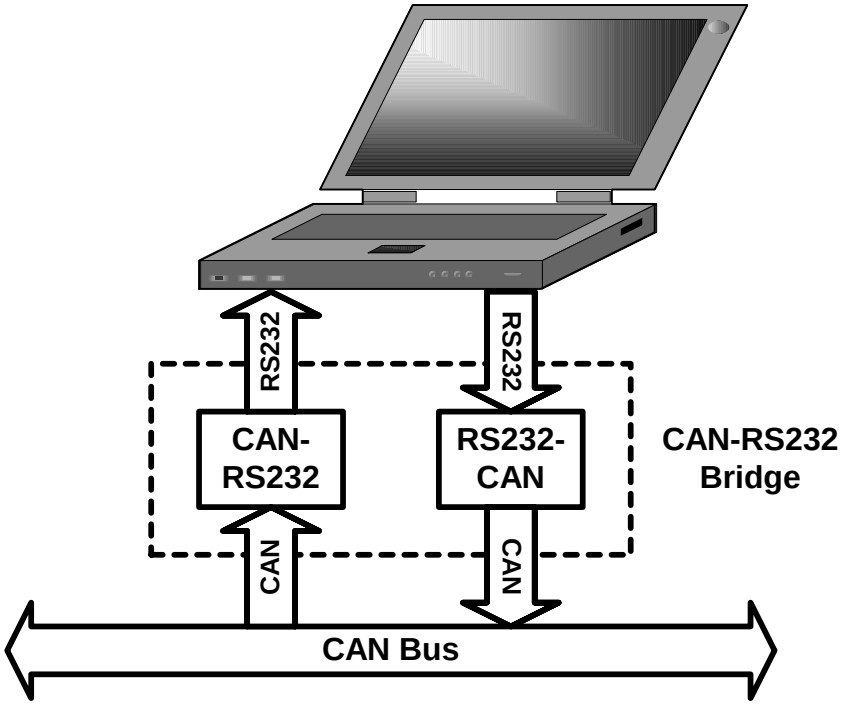


Figure 11. CAN to RS232 Bridge Node

Once all physical design requirements were met, a generic code set was developed to initialize and test the CAN-based system. For initial testing, nodes were either receive or transmit nodes. Future work combined the two functions into a fully operational CAN node.

The first initialization step was to configure the bit timing requirements for CAN message transmission. A bus rate of 250 Kbits/sec corresponds to a bit rate of 4 μsec. The CAN 2.0B protocol breaks each CAN bit into four sections (Figure 12). The SYNC_SEG is used to synchronize the bus nodes. The PROP_SEG compensates for physical delay times inherent within networks. PHASE_SEG1 and PHASE_SEG2 compensate for edge phase errors and can be lengthened or shortened by resynchronization.

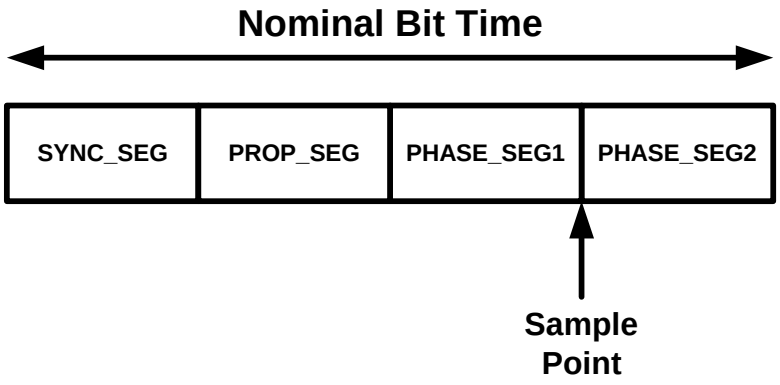


Figure 12. Nominal Bit Time (Reprinted from CAN 2.0B Protocol)

Each segment of the nominal bit time can be subdivided into several segments known as time quantum (tq). ISO 11783 recommends the time duration for each time state as shown in Figure 13.

tq = 250 nsec

SYNC_SEG = 0 tq

PROP_SEG + PHASE_SEG1 = 13 tq

PHASE_SEG2 = 2 tq

SJW = 1 tq

Total Bit Time = 13 tq + 2 tq + 1 tq = 16 tq = 4 μsec

Figure 13. Recommended Nominal Bit Timing

The Synchronization Jump Width (SJW) may result in either PHASE_SEG1 or PHASE_SEG2 becoming lengthened by 1 bit. The value for the SYNC must be 0 because the synchronization takes place only on the edge of a recessive to dominant transition.

The exact values recommended in ISO 11783 for each bit segment cannot be used in this project because the chosen oscillator speed of 20 MHz cannot be prescaled to give and exact time quantum output of 250 nsec. Instead the values shown in Figure 14 were used.

tq = 400 nsec

SYNC_SEG = 0 tq

PROP_SEG + PHASE_SEG1 = 7 tq

PHASE_SEG2 = 2 tq

SJW = 1 tq

Total Bit Time = 7 tq + 2 tq + 1 tq = 10 tq = 4 µsec (250 Kbits/sec)

Figure 14. Adjusted Nominal Bit Timing

The PIC18F258 contained three control registers associated with defining the CAN bit times. The three registers were configured as shown in Figure 15 to implement the appropriate bit times. These three CAN baud rate registers are standard for most CAN-based microcontrollers, independent of brand (CAN-CIA).

BRGCON1 = %00000011

BRGCON2 = %10100001

BRGCON3 = %00000001

Figure 15. PIC18F258 CAN Baud Rate Configurations

The PIC18F258 utilized two pins for CAN communication, RB2 (CAN transmission) and RB3 (CAN reception). The Data Direction Register should be appropriately set for each pin to allow data output from RB2 and data input from RB3. This was accomplished by using the TRIS command as shown in Figure 16.

TRISB = %00001000

Figure 16. Data Direction Register Configuration for CAN Messaging

The final node configuration required before CAN messages could be sent and received was to set the appropriate message receive filters within the receiving nodes.

When an incoming message was received by the CAN protocol engine, it automatically checked the message identifier against the values stored in the receive filter control registers. The receive filter must be set in software in order for messages to be received. There were four registers (32 bits) associated with each receive filter, which made up the 29-bit identifier used by CAN 2.0B. There were also masks associated with each filter that were required to be set in software.

Initial testing validated the ruggedness and repeatability of the CAN based control system. Four ECUs were linked together over the CAN bus and programmed to transmit data at a rate of 0.5 Hz. A Garmin (Kansas City, MO) 76 GPS receiver was used in simulation mode to provide NMEA strings while indoors. Potentiometers were used to simulate the feedback from the electronic actuators. Three nodes where designed to read independent analog voltages from potentiometers attached to each node. The potentiometer was connected as a single ended input directly to pin A0 on each of the nodes. A simple code (Figure 17) was implemented on each node to collect analog input data and transmit the measured values on the CAN bus.

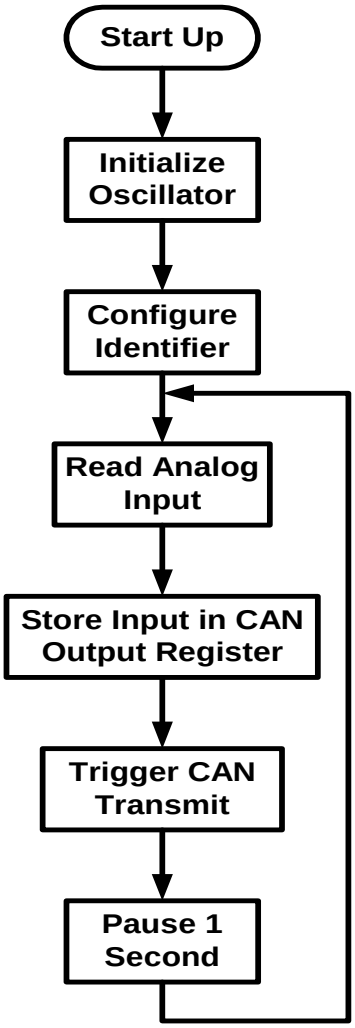


Figure 17. Initial Testing Program Flow Chart

The GPS node was unique in that the data were acquired through the on-chip serial port rather than the analog to digital converter. There was also no Pause statement for the GPS node. Its 0.5 Hz timing was based on the GPS output from the GPS receiver.

The message transmissions were logged via the CAN-RS232 bridge onto a task computer, thus modeling the final system. The incoming data messages were collected and time-stamped by an executable file created with Visual Basic.

The four transmitting ECUs and one CAN-RS232 bridge were tested for 24 hours and were subjected to random adjustments of the input voltages (Figure 18). The data messages were post-processed to determine the number of transmission errors that occurred within a 24 hour period. No transmission errors were found to have occurred

over the 24 hour period. The resolution of the stored data was only 2 seconds, so it was possible that transmission errors did occur, but that they were recognized by the ECU's and the errant messages were retransmitted according to ISO 11783 protocol.

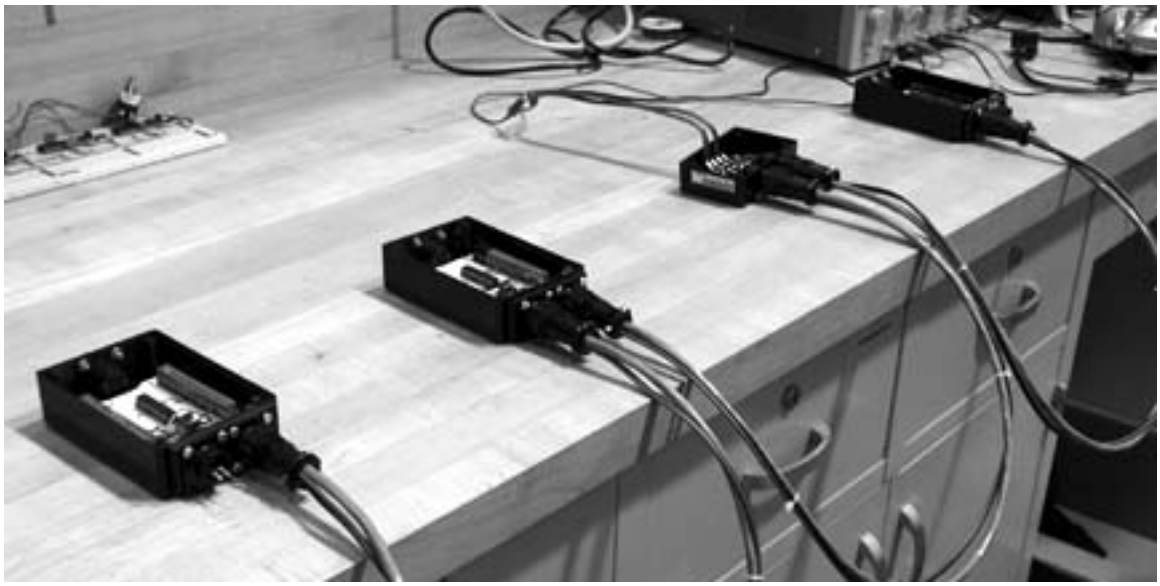


Figure 18. Laboratory configuration for Initial Node Testing

A secondary test was performed to determine the robustness of the CAN bus to external noise sources. A radio controller, similar to those used in remote aircraft flight, was placed adjacent to the CAN bus network cabling. An oscilloscope was used to monitor the noise level on the bus. The oscilloscope showed a tremendous amount of noise occurring on both CAN bus wires, but the messages still transmitted without errors (Figure 19).

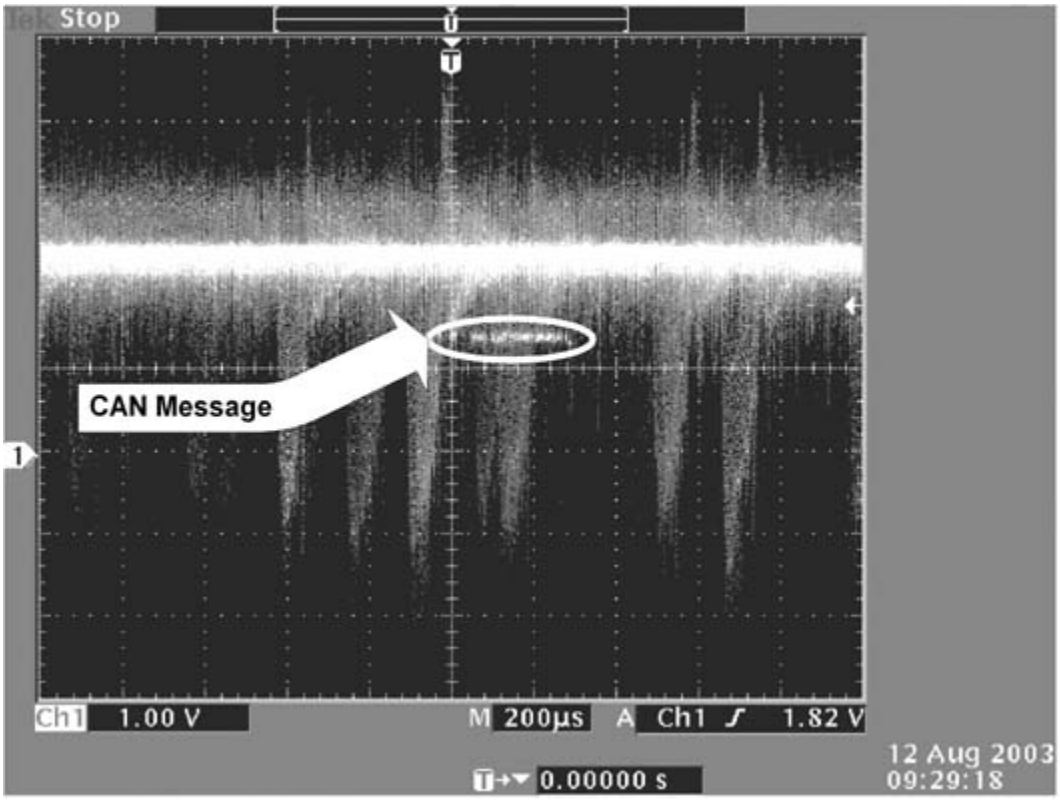


Figure 19. CAN_L Communication Line during High Noise Conditions

B. Sensor Interfacing

With the initial testing of the modular network completed, the next project objective was to interface each of the ECUs with sensors specific to autonomous vehicle guidance. This involved developing a list of sensor requirements to provide an appropriate level of functionality to the vehicle. The test vehicle for this project was an 18.6 kW hydrostatic drive tractor (Figure 20 &Figure 21). Manual linkages were used to actuate the steering axle, transmission speed, and three-point hitch location. All driver amenities were removed to reduce the gross vehicle weight and to lower the vehicle center of gravity.

Figure 20. Test Vehicle for Autonomous Guidance

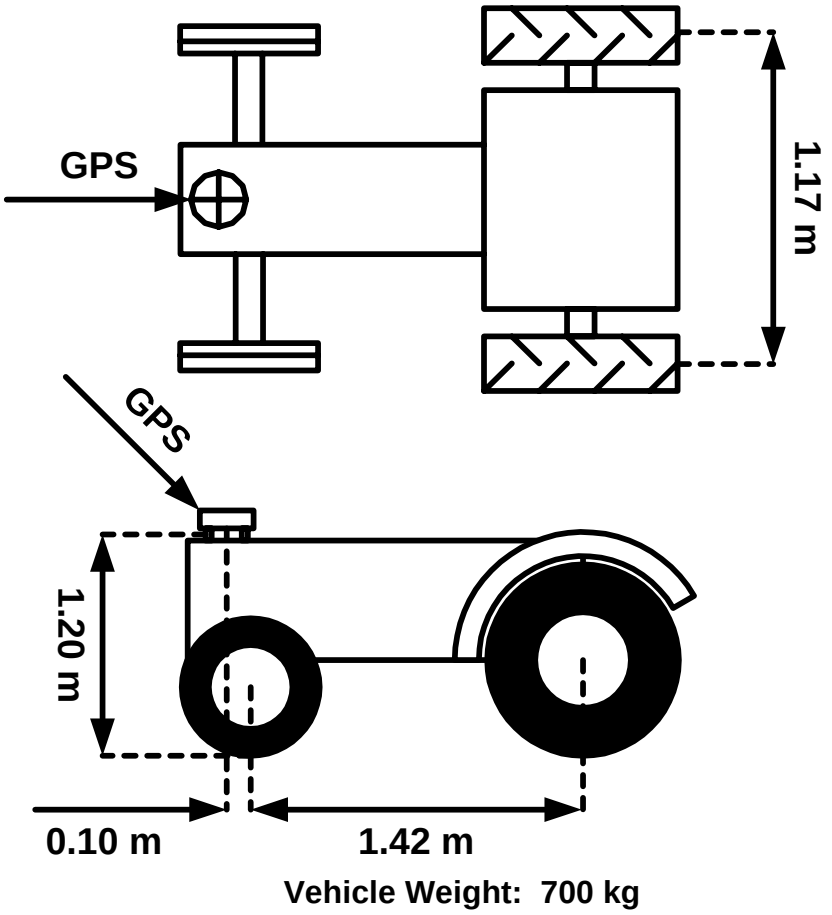


Figure 21. Test Vehicle Specifications

The following list outlines the minimum required control and sensing points to convert the test vehicle from manually operated to a limited functional capability autonomous machine.

1. Electronic control of the steering axle with position feedback
2. Electronic control of the hydrostatic transmission swash-plate with position feedback
3. Electronic control of the three-point hitch control lever with position feedback
4. Acquisition of vehicle position from a GPS receiver
5. Acquisition of vehicle heading from digital compass
6. Acquisition of vehicle turning rate from an analog gyroscope
7. Acquisition of control commands from a wireless RF controller.

The linkages on the vehicle that controlled the hydrostatic transmission swash-plate position, three-point hitch control lever, and front end steering were replaced with electric actuators (Figure 22). Each actuator contained a 10k potentiometer to provide position feedback. Each ECU responsible for executing a control command via an electronic actuator was interfaced with the position potentiometer through the internal analog to digital converter on the microcontroller. The supply voltage to the potentiometer was sourced from the node. By providing the supply voltage from the same source as the microcontroller, a single ended analog to digital conversion could be used with little worry of bias resulting from different ground potentials. Initial calibrations were performed to determine the maximum operating ranges for each of the potentiometers.

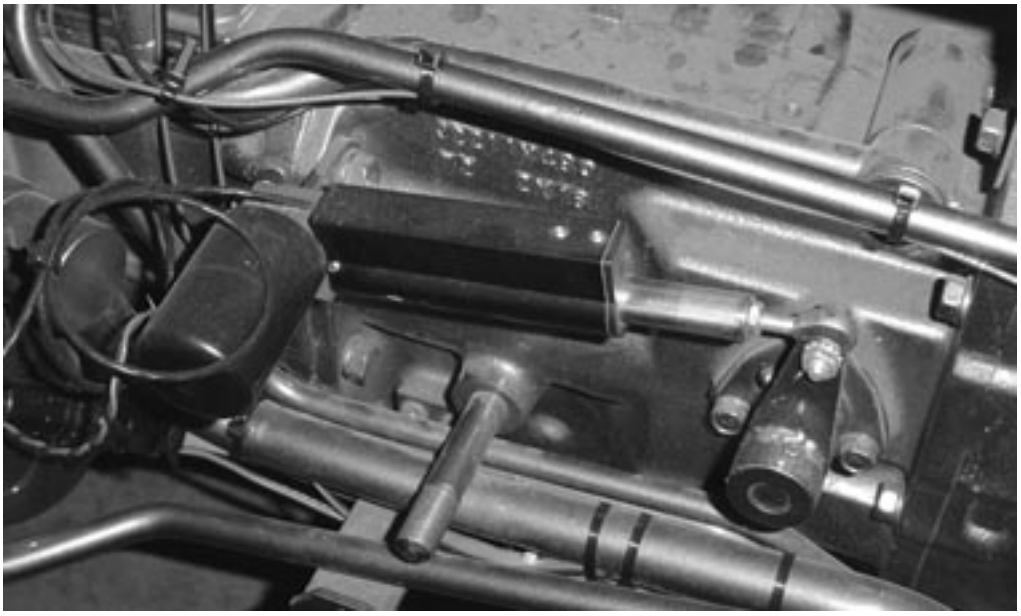


Figure 22. Electronic Control Actuator with Feedback Potentiometer

To calibrate the steering axle, a visual estimation was used to determine the center steering point and the extreme conditions. These locations were related to analog to digital converter count values and recorded to act as limiting values for the control program. The three-point hitch actuator was calibrated for two positions: maximum upward and downward position. The transmission actuator was calibrated for three positions: maximum forward, maximum reverse, and neutral. The three-point hitch actuator and the transmission control actuator did not require additional calibration, because their outputs were not critical to the goals of this project. In fact, the only critical point on either actuator was maintaining a neutral position on the transmission actuator in order to stop the vehicle. The steering actuator calibration was extremely critical and required a more sophisticated calibration method. Before this calibration could be accomplished, a GPS receiver interface was required.

A Trimble (Sunnyvale, CA) AgGPS 214 GPS receiver with WAAS correction provided vehicle positioning with sub-meter accuracy. The AgGPS 214 transmitted a vehicle position message in the CAN 2.0B format specified in SAE J1939. This message was not used for this project because of data processing limitations inherent to 8-bit microcontrollers. Specifically, the microcontroller and compiler combination could not process variables larger than 16 bits. The SAE J1939 standard format

packaged both latitude and longitude as 4 byte variables. To overcome this problem, an ECU was designed to monitor the NMEA output of the receiver via an RS232 connection and capture the GGA data string at a rate specified by the task computer. The latitude, longitude, time, GPS quality indicator, and number of satellites in use were each transmitted onto the CAN bus upon reception. The specific identifiers associated with each message are listed in Appendix A. The microcontroller source code for the GPS node and a detailed description of operation logic are listed in Appendix B.

The steering actuator was then calibrated to relate the count value acquired from the analog to digital conversion performed on the steering feedback potentiometer to the specific steering angle of the vehicle. The vehicle was subjected to a series of tests in which the steering position was held constant at a known feedback setting. While traversing a circular path, GPS location points were acquired by the task computer through the CAN-RS232 bridge and logged into a text file. The vehicle completed three circular paths for each steering position tested. The test was performed on a firm sod surface to reduce the effect of wheel side slip. The data were post-processed in a GIS package to determine the mean diameter of each circular path (Figure 23).

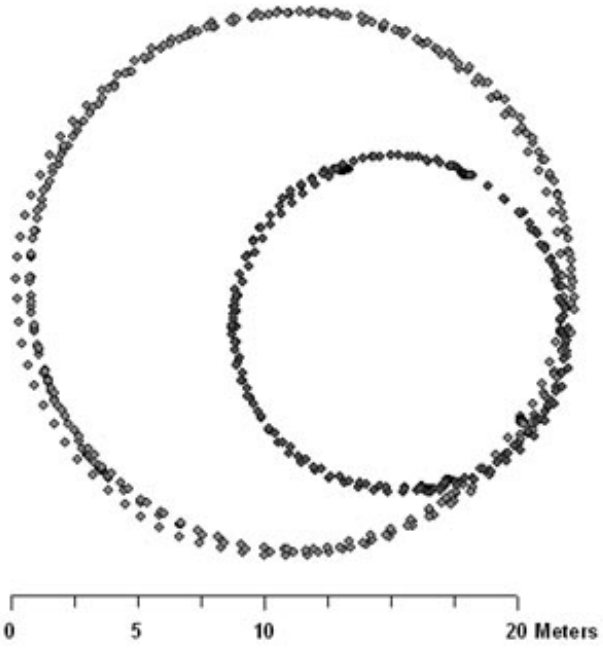


Figure 23. Steering Actuator Calibration

A kinematic model was used to define the vehicle's steering angle based on the radius of the circular path (Grovum and Zoerb, 1970). The model showed that the radius of each circle was directly related to the steering angle of the vehicle (Equation 2).

$$\phi = \arctan\left(\frac{WB}{R1}\right) \qquad (2)$$

Where: *R1* = effective turning radius of the vehicle

WB = vehicle wheel base

From Equation 2 and the multiple circular paths conducted during the calibration, an equation relating the actuator feedback counts to the actual steering angle of the vehicle was determined. This equation was as follows:

$$P = 19.70\phi + CenterPosition \qquad (3)$$

Where: *P* = Actuator Position (Counts)

ϕ = Steering Angle in Degree

For these tests, the value of CenterPosition was 470. Re-calibration was required due to drift in the steering axle position potentiometer. This number required regular adjustment throughout vehicle testing based on new front end calibrations. A reference calibration line was etched into the actuator as a way to quickly re-identify the center position. A linear regression was performed to determine the accuracy of the steering angle equation (Figure 24). The R^2 values for the linear regressions were .998 and .999 for counterclockwise and clockwise turns, respectively.

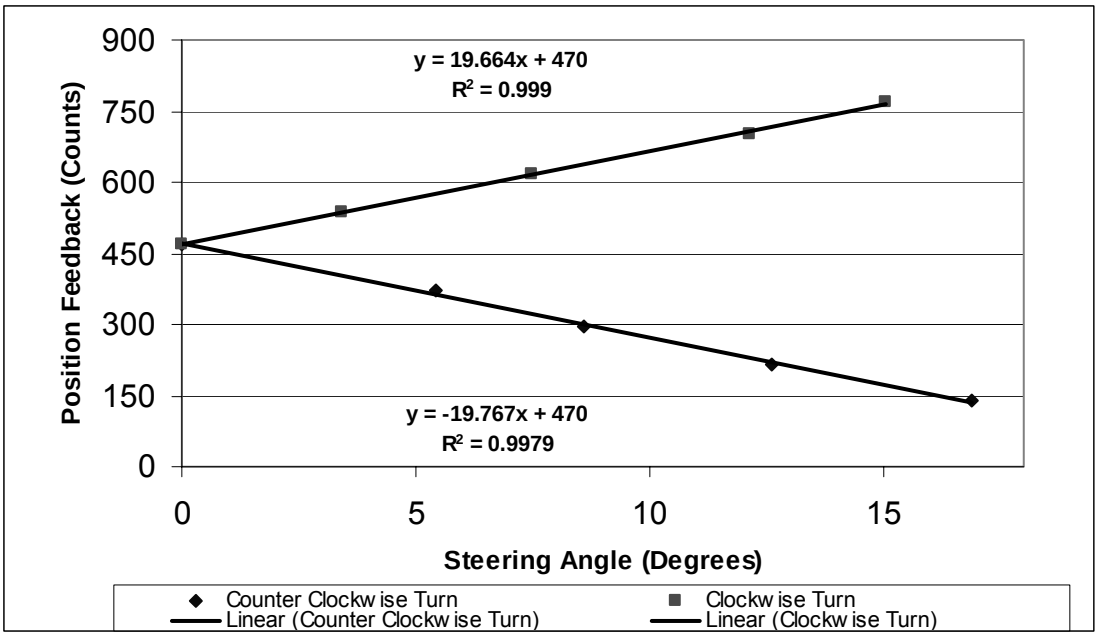


Figure 24. Steering Axle Actuator Calibration

Each actuator used in this design was a ball-screw linear actuator. Due to the large current requirements of each actuator, the microcontroller itself could not supply the necessary power to operate the actuator motor. A separate circuit was required to operate the electronic actuators remotely from the ECU. An H-bridge motor controller was installed to isolate the ECU from the high current and voltage loads necessary to drive the actuators. An LMD18201 H-bridge produced by National Semiconductor (Santa Clara, CA) was used for the transmission and hitch actuators. This H-bridge could supply up to 3 amps of continuous current at up to 55 volts. The LMD 18201 did not have sufficient current capabilities for the steering actuator, which could draw up to 15 amps continuously. For this application an OSMC3 (www.robot-power.com) motor controller was used. This controller could provide well in excess of 15 amps continuously. Digital output pins on the ECU controlled the H-bridge circuits and allowed forward and reverse motion and locking of the electric actuators. Initial implementation did not utilize the locking capabilities of the H-bridges. It was found that the actuators would overshoot the desired location due to the inertia built up in the motor and ball-screw mechanism. H-bridge braking was then incorporated into the ECU control routine, which dramatically reduced the magnitude of actuator.

As stated in the requirements, a means to determine the current vehicle heading direction was required. Several methods were available to acquire this value including computing a value based on the incoming GPS parameters. The most practical solution to the problem was to utilize a simple board level digital compass. This allowed for rapid sampling of the heading state as opposed to the limited sampling capabilities of the GPS receiver. It also allowed for much easier implementation because the microcontroller could interface the compass through a simple serial communication link.

A Vector 2X digital compass produced by Precision Navigation (Santa Rosa, CA) was chosen and interfaced with the steering ECU. The steering ECU was chosen as the interface node because of its proximity to the front of the vehicle and it potential use for heading information. The sensor was a low-cost, 2-axis compass and magnetic sensor that provided accuracy of 2° for vehicle heading at a resolution of 1˚. The sensor utilized a patented magneto-inductive magnetometer. The sensor was calibrated to remove hard iron distortions in the form of a constant offset that resulted from the host vehicle. Non-constant magnetic fields from objects such as AC motors could not be calibrated out of the system and were physically avoided. Additional noise suppression was incorporated by mounting the compass 12” in front of the forward most part of the vehicle on a non-magnetic wooden surface.

The sensor was interfaced to the steering node via an SPI communication link. Five data lines were required to interface the compass, which was the slave in the system. Two data lines initialized the sensor and forced it to sample the heading direction. Three more lines (chip select, clock, and data) were then used to receive the heading information into the microcontroller. The compass could provide data at a maximum rate of 4 Hz. The source code used to access heading data from the compass can be reference in Appendix C.

The current turning rate of the vehicle was desired to provide an auxiliary feedback mechanism to the steering ECU. It was hypothesized that this sensor could sense problems with the steering axle calibration or with vehicle guidance problems due to inadequate ground traction and front wheel sideslip. This would be accomplished by comparing the actual turning rate of the vehicle to the predicted turning rate as defined by a kinematic model.

A CG-16DB0 piezoelectric ceramic gyroscope produced by NEC Tokin (Union City, CA) was incorporated into the system to provide real time turning rate feedback. This sensor was comprised of a single piezoelectric ceramic column printed with electrodes that provided a sensitivity of 1.1 mV/°/sec with a sensing range of ±90 °/sec. It was commonly used in stability controllers for small radio-controlled aircraft. Although a much more expensive gyroscope could provide much greater accuracy and most likely be much easier to interface, the gyroscope was the lowest priority sensor required by the vehicle and little resources were invested to it.

The Ceramic Gyro voltage output was connected to an instrumentation operational amplifier circuit to increase resolution for turning angles from 0 °/sec to 10 °/sec. The output signal was amplified by a gain of 450, resulting in the calibration curve seen in Figure 25. A dedicated ECU was developed to read the conditioned output from the Ceramic Gyro and over-sample the signal by a factor of 25. Thus, 25 continuous readings were averaged and transmitted as the current turning rate. The sensor was located just behind the midpoint of the rear axle on the vehicle. This location was very near the point of rotation of the vehicle while traversing a headland turn. The sensor was mounted on a vibration isolation surface to limit the effects of mechanical vibrations within the vehicle on sensor output.

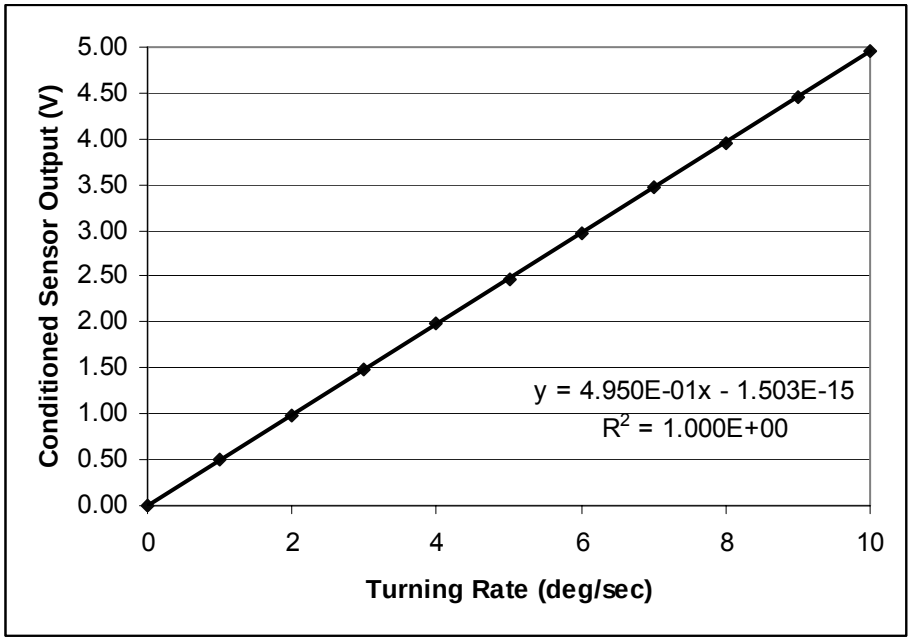


Figure 25. CG-16DB0 Conditioned Calibration Curve

Field testing was performed to determine how the sensor would respond to the turning rate change of the vehicle during normal operations. The vehicle was guided through a standard row and headland turn operation. Theoretically, the sensor should have responded to the change in turning rate while the vehicle was traversing the headland turns. Unfortunately, during field testing, the Ceramic Gyro reported erratic results directly related to the mechanical vibrations of the vehicle. Several attempts were made to isolate the sensor mount from these vibrations, but all designs proved inadequate. These alterations included increasing the amount of damping material located on the sensor mount and also implementing an analog passive filter circuit to reduce the amount of high frequency noise. The Ceramic Gyro was no longer incorporated into the system design because of the mechanical vibration noise.

The final requirement of the interface system was to transfer signals from a radio-controller onto the CAN bus. The radio-controller ECU monitored four channels from the RF receiver: variable steering angle, variable transmission speed, three-point hitch state, and vehicle enabled state. Each of these channels were monitored and transmitted on the CAN bus at a rate of 25 Hz. For the two variable controls (steering and transmission), the ECU measured the pulse on-time of the corresponding input channel with a resolution of 2 μsec. A calibration equation was developed by relating the maximum and minimum pulse on-times from the receiver to the appropriate maximum and minimum count values for each actuator. Figure 26 shows the calibration curve for the steering actuator. No calibration curve for the transmission is presented because the calibration equation changes depending on the vehicle throttle setting. The vehicle speed was calibrated before each test to ensure consistent test methods. For the three-point hitch and vehicle enable channels, a two position pulse input was read from the RF receiver (Figure 27). In the high on-state, the three-point hitch was raised to its highest point; during the low on-state, the hitch was lowered to its lowest position. The height of the three point hitch was measured from ground level to the pin location at the end of the lower lift arm. The vehicle enable channel was measured in a similar manner, although if a low on-state was detected, messages were sent to all control nodes to return to the neutral position and stop the vehicle. Further detail of the node operation can be found in the source code located in Appendix C.

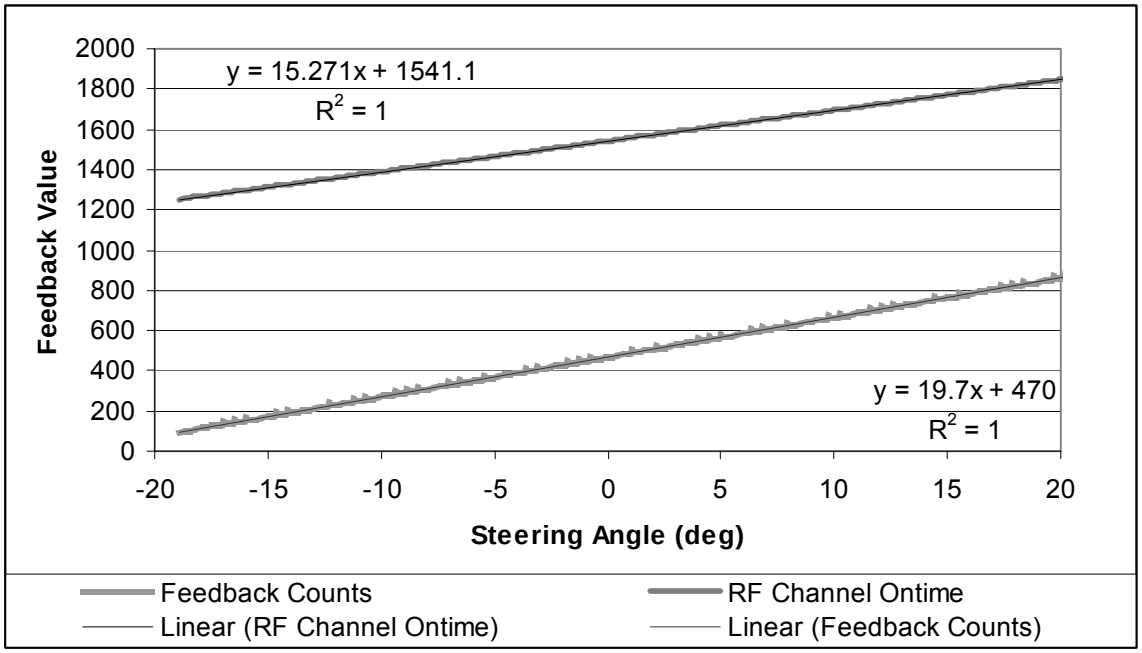


Figure 26. RF Controller Calibration Curve for the Steering Actuator

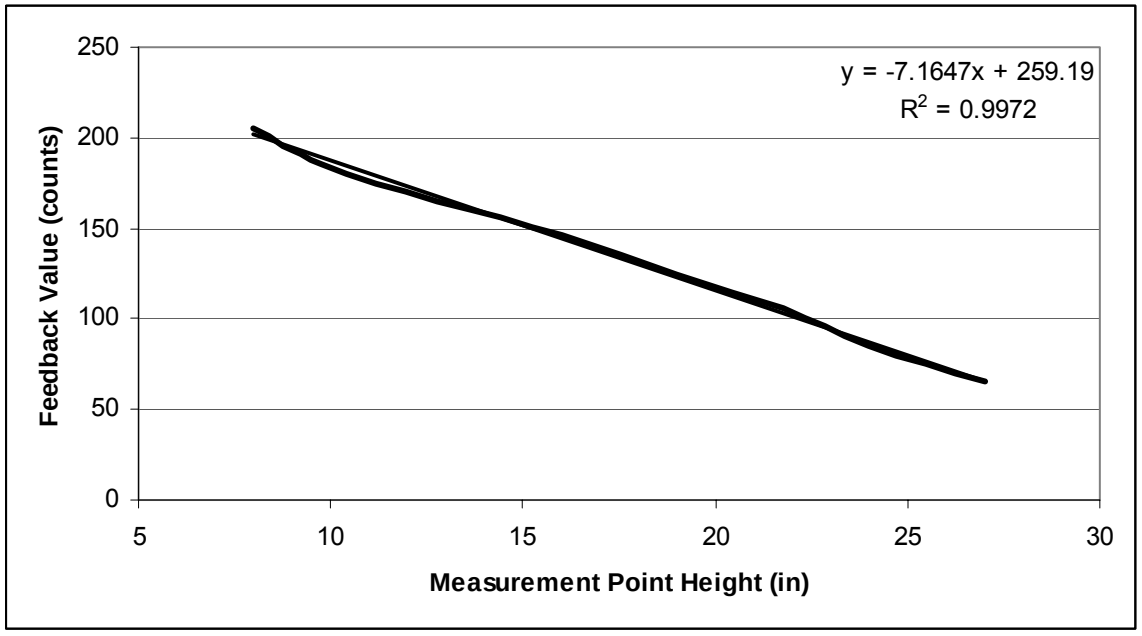


Figure 27. Calibration Curve for the Three Point Hitch

Figure 28 depicts the complete network layout used to satisfy the system requirements.

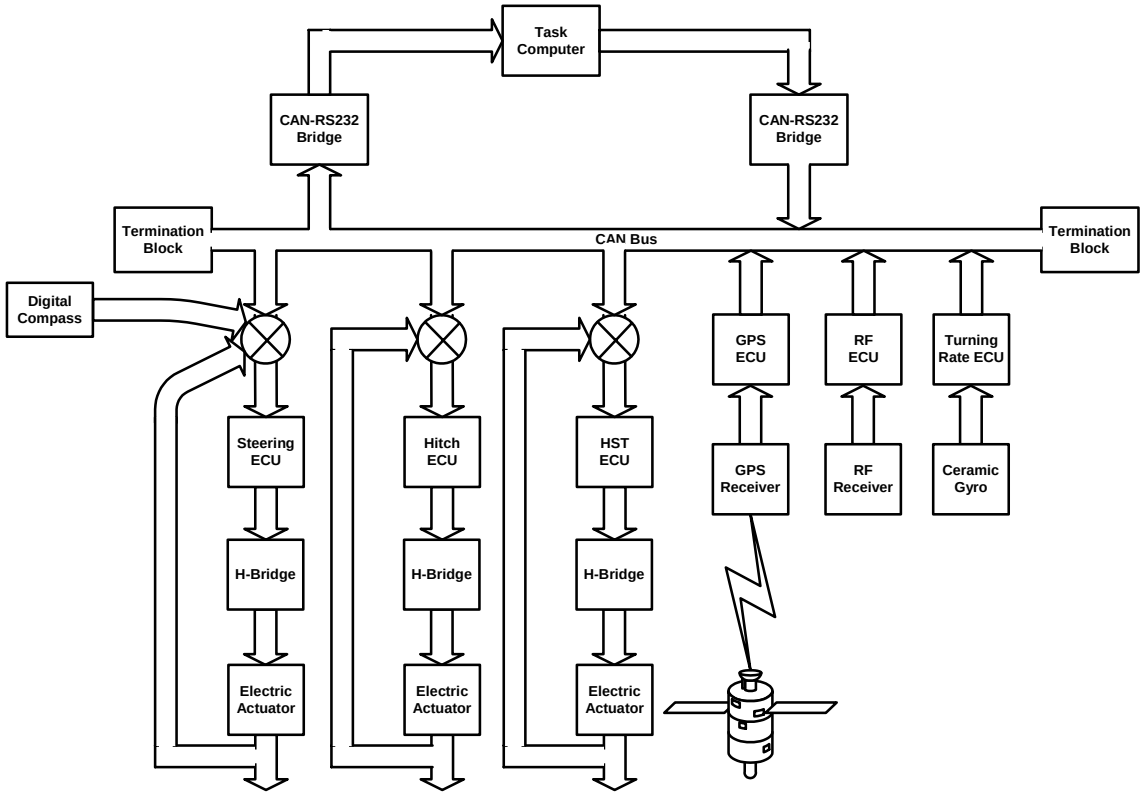


Figure 28. Final Network Layout

Since the steering, three-point hitch, and transmission nodes were the actuation points for the distributed control system, they were required to accept a control set-point and implement a feedback control routine to correctly position the actuator. The ECU was programmed to continuously run the same control routine, thus it monitored the CAN receive buffers for a new set-point message and conducted a feedback control routine to maintain the actuator at the desired location (Figure 29). The source code for this operation can be referred to in Appendix C.

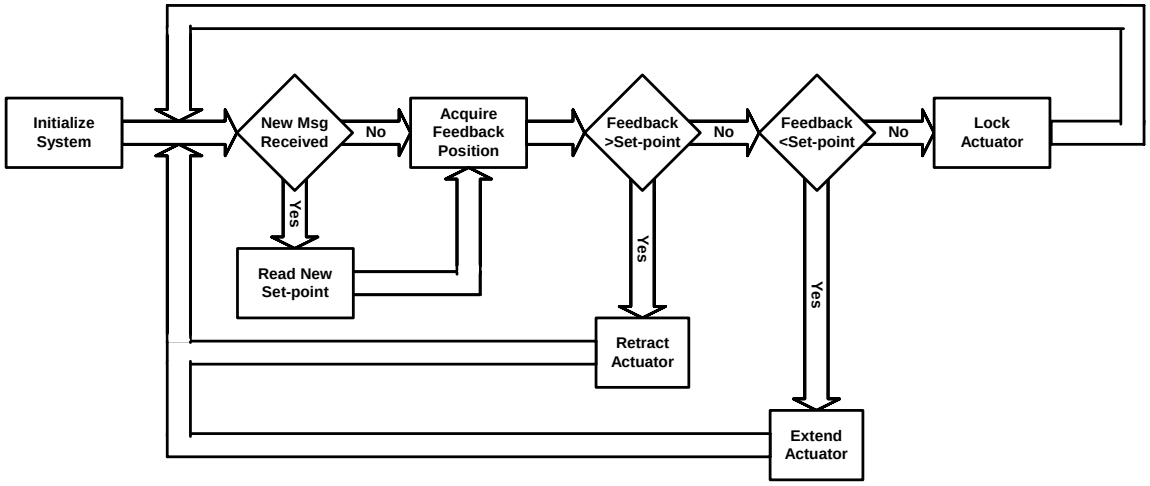


Figure 29. ECU Control Routine

To test the feedback operation of the ECUs, control commands were asserted on the bus from the task computer manually. It was noticed that the steering ECU had a difficult time positioning the actuator exactly on the desired location. The ECU was determining the current position via a 10-bit analog to digital converter. A noise level or fluctuation in the feedback potentiometer of 4.9 millivolts or higher would result in a change in the perceived actuator position and cause the algorithm to correct for this change. The result was an excessive control effort that led to oscillations in the output. To combat this problem, a 1% deadband was programmed into the control algorithm to allow for slight variation in the feedback signal and noise within the system. The 1% deadband was calculated based on the full scale range of the sensor. Thus, for all five volt potentiometers used for the control nodes, the deadband was ± 5 counts.

The H-bridge controllers used for this project did have the capability to receive an analog input signal and produce a proportional output control to the actuator motors. It was found that the control system reacted with sufficient speed such that it was not necessary to implement this feature. The final control system for the ECU controllers was then a Bang-off controller.

C. Development of a Guidance Algorithm for Autonomous Vehicle Control

Each of the sensors and nodes depicted in Figure 28 were installed on the test vehicle. CAN bus messages were defined to efficiently transfer feedback data into the

task computer and transmit control commands to ECUs. All messages required for autonomous vehicle operation were sent each time a new GPS message was received. Two different GPS update rates were used for this project: 1 Hz and 2 Hz. Average bus load during normal transmission with a bus speed of 250 kbits/sec, a GPS signal rate of 2 Hz, and an average message length of 150 bits was 0.72%. This was very desirable and allowed for extensive expansion of the system.

The vehicle ground speed during autonomous testing was set at 3 km/hr (1.86 mph), which resulted in a vehicle travel distance of 0.467 m (1.37 ft) per control message when operating at a 2 Hz update rate. Appendix A details all messages required to control the vehicle during autonomous operation. The identifier priority levels used for this project correspond to the message priority levels defined in ISO 11783 Section 3 (Table 3).

The first problem encountered during initial setup and testing was that the CAN transceiver chips were failing at a high rate. Discussions with the manufacturer determined that this failure was most likely caused by the noisy input voltage from the vehicle's battery. Specifically, the noisy voltage caused the current draw through the system to exceed the specifications of the transceiver chip. A solution was reached by adding another voltage regulator to create an isolated power supply for the transceiver in each ECU. While this did reduce the occurrence of transceiver failures by roughly 90%, they still did occur. All transceiver chips were acquired in one shipment before the parts were in full production. It is possible that there may have been lingering flaws in the manufacturing process or chip design that had not been fully alleviated.

With the CAN-RS232 bridge installed, a task computer was required to link the CAN based control system with a process control system. A Pentium 4® 1200 MHz laptop computer served as the task computer. The task computer ran an executable file that was written and compiled using Microsoft (Redmond, WA) Visual Basic 6.0®. The program received information from the CAN bus through a CAN-RS232 bridge node. Incoming messages were decoded based on the source address of the sender and used for vehicle guidance (Figure 30).

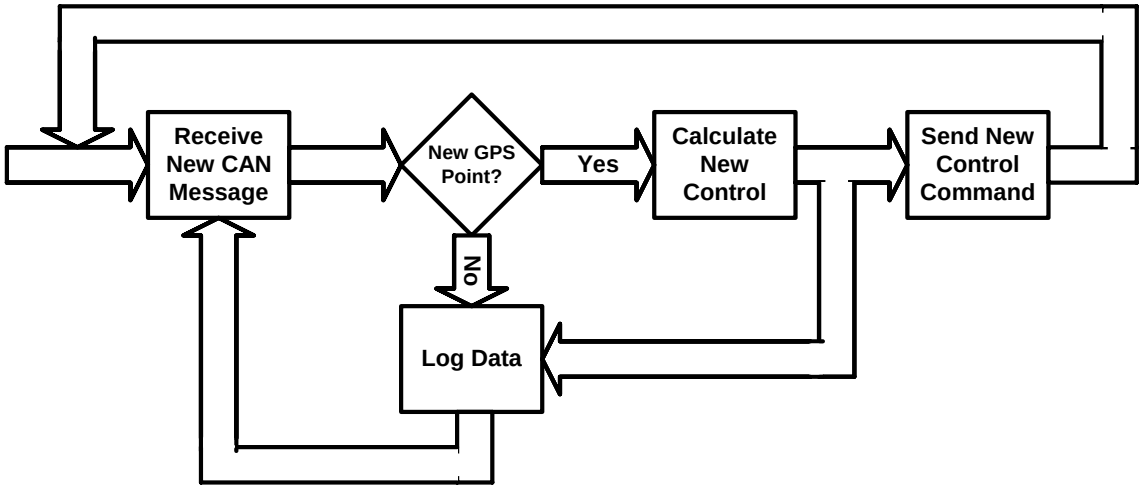


Figure 30. Task Computer Flow Chart

The implementation of automated straight-line guidance control by the task computer can be broken into five specific steps.

1. Initialize the system by defining a desired path of travel
2. Receive feedback information from the GPS receiver through the CAN bus
3. Determine the appropriate steering axle position for autonomous guidance
4. Transmit the appropriate control commands to the steering and transmission ECU
5. Log position and control data for post-processing and evaluation.

To accomplish the first step, the task computer was given initial geographic Points A and B. The task computer created a line between the two defined points and made guidance decision based on this desired line of travel. These two points, as well as all subsequent locations received from the GPS receiver entered the task computer in WGS 84 format and were projected to Universal Transverse Mercator (Zone 16N) Cartesian coordinates. The WGS 84 format contained ten significant digits, thus a resolution of 1 mm was available after the UTM conversion. The task computer received the GPS message by matching the GPS identifier with the identifier of the incoming message. The specific GPS message parameters are detailed in Appendix A.

Each time a new GPS message was received, the task computer calculated the current vehicle error from the desired line of travel and applied a digital control routine to determine the new guidance control command. This command was transmitted through

the CAN-RS232 bridge and was received by the steering node. A transmission control command was also transmitted at the same time, but simply directed the transmission to maintain a constant forward ground speed of 3 km/hr. During the latency time between incoming GPS messages, the task computer recorded position and control parameters to a hard disk for future evaluation. The GPS locations were stored with their appropriate time stamp, so no latency was seen in the recorded data. Each set of guidance parameters was stored in a comma-delineated text file that ended with a linefeed and carriage return. The data were then easily accessed by commercially available spreadsheet and database software packages. Complete detail on the operation of the task computer can be acquired from Appendix B.

i. Kinematic Model Based Guidance Controller

For the task computer to effectively control the test vehicle during autonomous operations, a control algorithm must be developed. Several models have been developed that predict the response of a tractor relative to changes in the steering angle. The test vehicle for this project was light weight, small, and slow, thus a purely kinematic model was used to describe the system. More specifically, the bicycle model described previously was applied to this test vehicle.

In order to apply a control algorithm, a control point or set point must be defined. The set point was defined as a point that lies on the desired path line at a specified distance ahead of the point on the path line perpendicular to the current vehicle location. This specified offset distance was defined as the look-ahead distance and was represented by the variable LD (Figure 31). The Look Ahead Distance was set at 5 m for this project, which corresponded to the magnitude of the step input used to test the vehicle response. Several look-ahead distances were considered, and while the decision to use 5 m was not totally arbitrary, there was no comprehensive reason driving its selection.

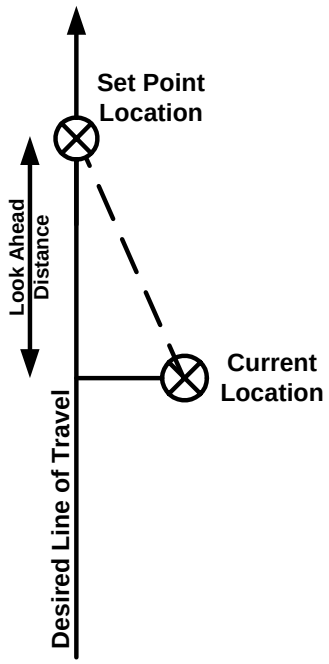


Figure 31. Look Ahead Distance Diagram

By assuming the vehicle was already heading along a line that was parallel to the desired line of travel, the desired steering angle was computed for the vehicle to pass through the set point. The kinematic model was used for this computation (Grovum and Zoerb, 1970). Equation 6 showed that the steering angle could be calculated based on a computed radius of curvature from the current vehicle position to the desired set point (Figure 32).

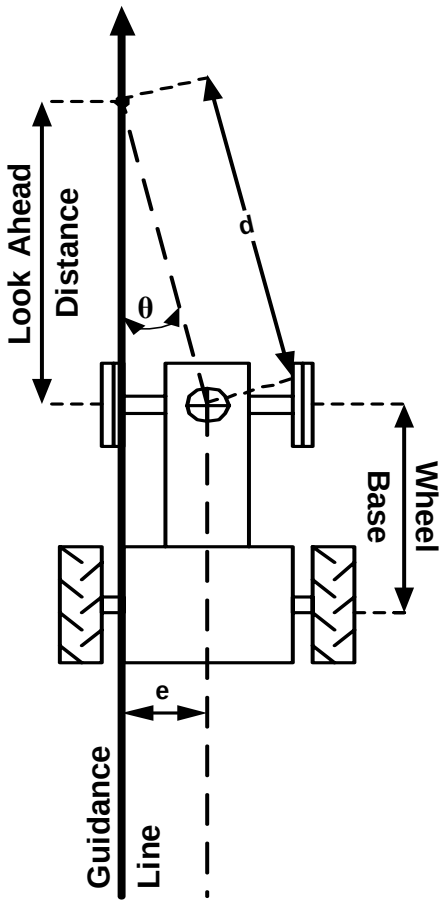


Figure 32. Kinematic Model of the Proportional Control Parameters

$$\theta = \arccos\left(\frac{LD}{d}\right) \quad (4)$$

Where: *LD* = Look Ahead Distance

d = Distance Between Current and Set Point

θ = Angular Deviation

$$R_{CP} = \frac{d}{2\sin(\theta)} \quad (5)$$

R_{CP} = Path Radius of Curvature

$$\alpha = \arctan\left(\frac{WB}{R_{CP}}\right) \quad (6)$$

WB = Wheel Base

α = Proportional Steering Angle

Equation 6 showed that the Look Ahead Distance directly affected the desired proportional steering angle. With this being the case, the Look Ahead Distance remained 5 m throughout the project and the proportional gain was tuned accordingly. Under normal operating circumstances, the proportional controller alone was not sufficient for the vehicle to settle on and track a desired path. As the vehicle approached the set point it traveled along a line that was no longer parallel to the desired path. This contrasted the assumption made during the development of the proportional control. A derivative control was needed to compensate for the difference between the desired vehicle heading and the current vehicle heading. The derivative control was developed based on the slope differential between the desired path and the actual path. The actual path slope was calculated by creating a line between the two most recent GPS location points. Again, the steering angle was calculated from the kinematic model based on a radius of curvature defined by the GPS points (Figure 33 & Equation 8).

$$R_{CD} = \frac{d}{2\sin(\mu)} \tag{7}$$

Where: R_{CD} = Derivative Radius of Curvature

d = Distance between Current and Set Point

μ = Slope Differential

$$\tau = \arctan\left(\frac{WB}{R_{CD}}\right) \tag{8}$$

Where: τ = Differential Steering Angle

WB = Wheel Base

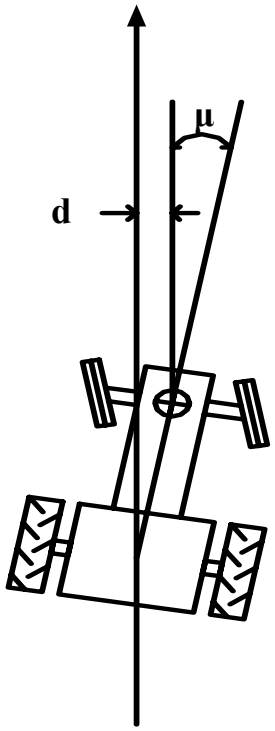


Figure 33. Kinematic Model of the Derivative Control Parameters

The angles calculated for the proportional and derivative control were incorporated into a control equation for the vehicle (Equation 9). Proportional and derivative gains were implemented to provide tuning capabilities to the system. The system was tested under a unity gain to create a baseline step response for the vehicle (Figure 34).

$$\phi = (P_G \times \alpha) + (D_G \times \tau) \quad (9)$$

Where: ϕ = Steering Angle

P_G = Proportional Gain

D_G = Derivative Gain

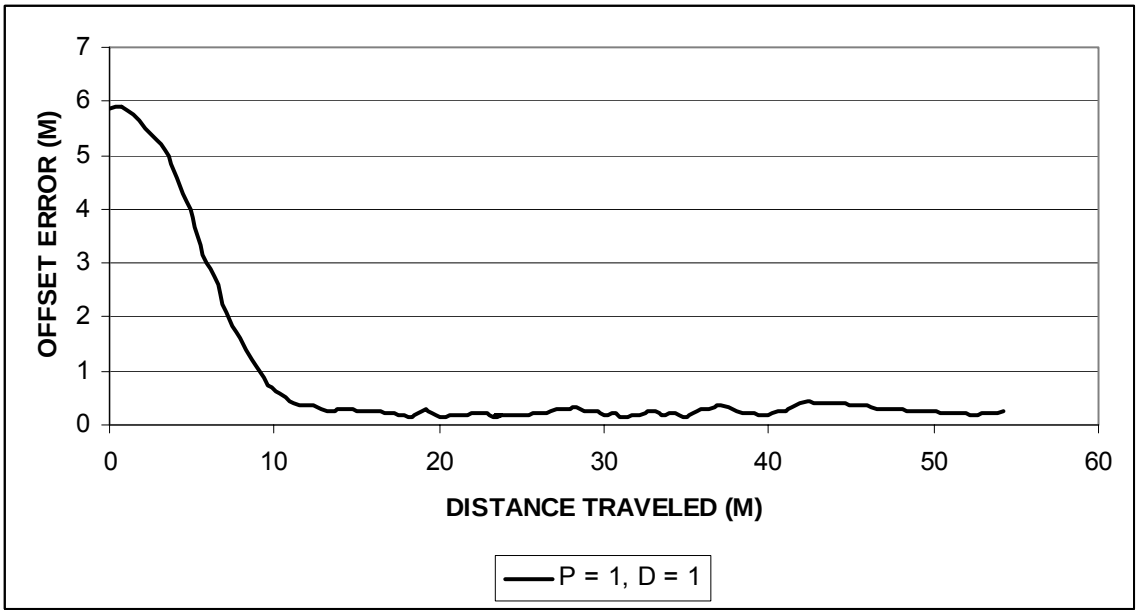


Figure 34. Step Response for Unity Gain on Sod Surface

Several conclusions were drawn from this unity gain test. The system was over damped as seen from steady state error and extended settling time in Figure 34. The proportional gain was increased to reduce the settling time. Also, the derivative gain was reduced to allow for a small overshoot during the step response. The sensing location was located at the front of the vehicle, so a minor overshoot will allow the rear end of the vehicle to converge to the guidance line faster.

A safe assumption from this first test was that the system can be modeled as a second-order system. If it was a higher order system, there was very little residual effect from the third and higher order system components. According to Doebelin (1998) a damping ratio (ζ) of 0.65 is often used for second-order systems that require a quick, yet stable response. During a five meter step response, this enabled an overshoot of 0.34 m. Based on the slope approach recorded during the unity gain test, an overshoot of 0.34 m will position the rear of the tractor very near the desired line of travel with no overshoot. Thus a damping ratio of 0.65 was used to tune the PD controller.

The PD controller gain settings were tuned through a series of field trials. Four proportional gains were tested ranging from 1 to 3 (Figure 35). It was found that a proportional gain of 1.5 provided a ζ value of 0.745. Since the proportional gain was set

very close to one, the bicycle model provided a sufficient representation of the vehicle motion. The derivative gain was tuned to obtain a ζ value closer to the desired value of 0.65.

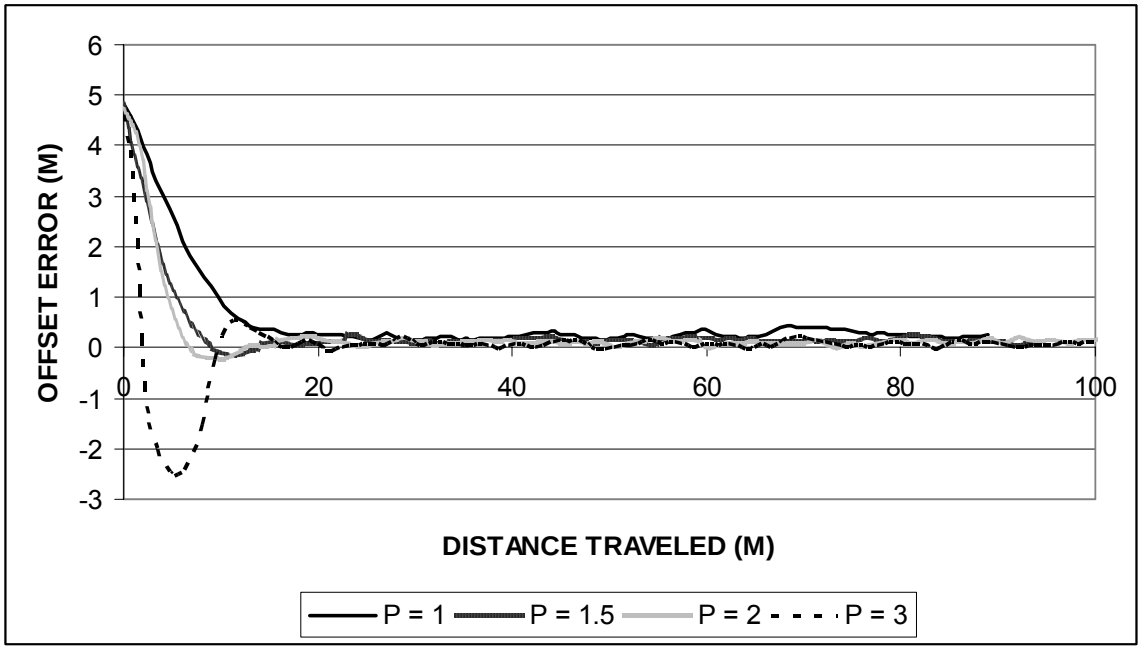


Figure 35. Step Response for Various Proportional Gains with a Constant Derivative Gain of One

Table 6. Step Response Characteristics for Proportional Gain Testing

	Proportional Gains			
	1	**1.5**	**2**	**3**
***Ds (m)*[1]**	17	8.5	10.4	14.7
***Mp (%)*[2]**	-	3.0	4.4	50.6
***Dp (m)*[3]**	-	11.5	10.5	6.3
***Dr (m)*[4]**	-	9.2	6.9	2.2

[1] *Ds* = Settling Distance

[2] *Mp* = Percent Overshoot

[3] *Dp* = Peak Distance

[4] *Dr* = Rise Distance

Further testing of the derivative gain was performed using a proportional gain of 1.5 (Figure 36). It was hypothesized before the tests that a lower derivative gain would reduce the damping in the system and thus reduce the settling time. A small reduction in the derivative gain should result in an improved damping ratio of very near 0.65.

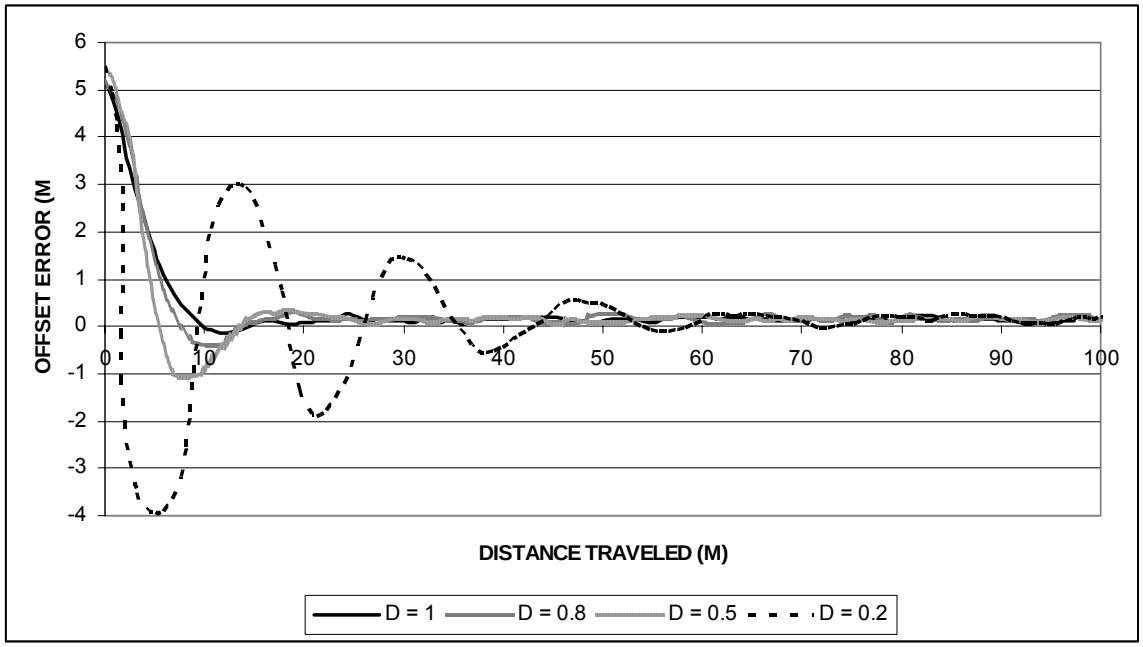


Figure 36. Step Response for Various Derivatives Gains with a Constant Proportional Gain of 1.5

Table 7. Step Response Characteristics for Derivative Gain Testing

	Derivative Gain			
	1	**0.8**	**0.5**	**0.2**
Ds (m)	8.5	14.2	19.4	51.4
Mp (%)	3.1	7.2	21.6	92.5
Dp (m)	11.8	10.4	7.8	5.0
Dr (m)	9.2	7.6	5.4	1.6

The step response performance was improved by decreasing the derivative gain slightly (Figure 36). A derivative gain of 0.8 provided an overshoot of 7.2%, which corresponded to a damping ratio of 0.64. This was ideal for the vehicle response; thus, a derivative gain of 0.8 was adopted.

Tests conducted using the PD controller revealed that the control system produced a steady state offset error that could not be reduced by further adjustment of the PD gain parameters. The magnitude of the steady state error was 0.154 meters. An integral control was incorporated into the system to reduce this steady state offset. A classical digital integral controller was analyzed and determined that it would not work for this particular situation (Equation 10).

$$u_I(k) = K_2[u(k-1)+Tx(k)] \quad (10)$$

Where:

$u_I(k)$ = Integral Control Command

K_2 = Integral Gain

$u(k-1)$ = Previous System Control Command

T = System Sampling Time

$x(k)$ = Current Error Signal

The classical integral control was based on the previous command and current offset error value. The GPS location points contain too much inherent noise and variability to calculate a reasonable steering correction. The previous tests showed that at least 9 position error points needed to be averaged to accurately predict the steady state offset. Again, this estimated offset error was used in conjunction with a look-ahead distance to determine a corrected steering angle based on the bicycle model. The formula for the integral controller was:

$$\gamma = \arctan\left(\frac{WB \times \sum_{-9}^{0}(OE_n)}{10 \times d_{ss} \times LD}\right) \quad (11)$$

Where:

γ = Integral Steering Angle

OE_n = Offset Error at Point n

WB = Wheel Base of the Vehicle

d_{ss} = Distance between Current and Set Point

LD = Look Ahead Distance

The new form of the PID implementation equation became:

$$\delta = P_g \times \phi + D_g \times \mu + I_g \times \gamma \qquad (12)$$

Where: I_G = Integral Gain

The new PID controller was tested to determine if the integral control could reduce the steady state error in the system. Integral gain values from 0 to 10 were tested. As the integral gain increased, the steady state error was reduced, but the standard deviation of the error increased (Figure 37). These responses were expected based on digital control theory. What was not expected was that a significant steady state error of at least 0.08 m continued to exist.

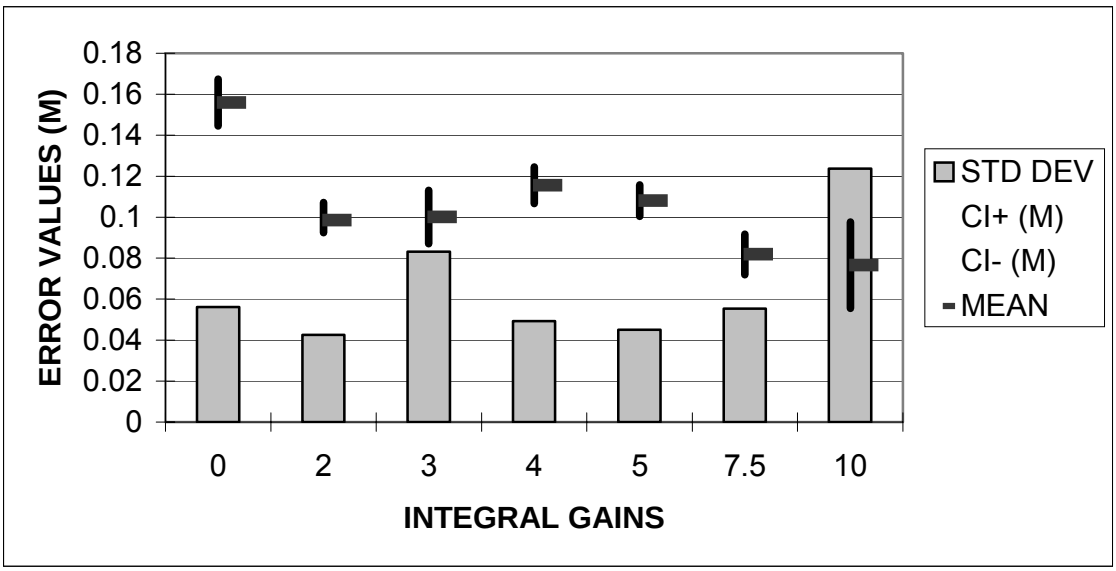


Figure 37. Comparison of Integral Gains for Steady State Error Reductions

The statistical parameters used to describe the accuracy of the control system were: mean error value, standard deviation of the error, and the mean 95% confidence interval. These statistical parameters were calculated during the steady state portion of the test. Steady state was defined to begin when the vehicle performance settled to within 2% of the input offset amplitude. An accurate system should produce a mean error value at or near zero along with a 95% confidence interval of no more than 5 cm. Julian (1971) showed that deviations of 5 cm or less at the front wheel produce negligible deviations at the rear wheel. The position points used for the error analysis were the current vehicle position, thus no software filtering was performed on this data.

The accuracy measurements were relative to the accuracy of the GPS receiver, thus they were not absolute accuracies.

Several factors could have been causing the steady state error to exist.

1. The control system implemented an overly complex algorithm to determine the appropriate control value
2. The GPS signal utilizing WAAS correction was supplying a noisy signal
3. The derivative gain parameter was too large when the offset error was small and caused impulsive control responses.

One of the system parameters logged by the task computer was the seconds value from the internal computer clock. The control algorithm was set to operate off the GPS signal input, thus it was expected that this clock value would be incremented sequentially for each control command. Inspection of the data revealed that often two control commands would occur within the same second cycle. This indicated that the program was struggling to execute all the required algorithms within the one second cycle between messages.

Closer examination of the error signal from the PD tuning determined that there were significant impulses within the GPS signal. This was expected due to the fact that WAAS correction was being used. There was also a noticeable lag between data points. This was not caused by the latency of the task computer, but rather by the fact that corrections occurred at a rate of 1 Hz.

The derivative gain term also negatively impacted the ability of the control system to squelch the steady state error. It was found that as the vehicle operated in the steady state region, the derivative control signal oscillated between a positive and negative value. This oscillation along with the 1 Hz update rate drove the vehicle away from the desired guidance path (Figure 38).

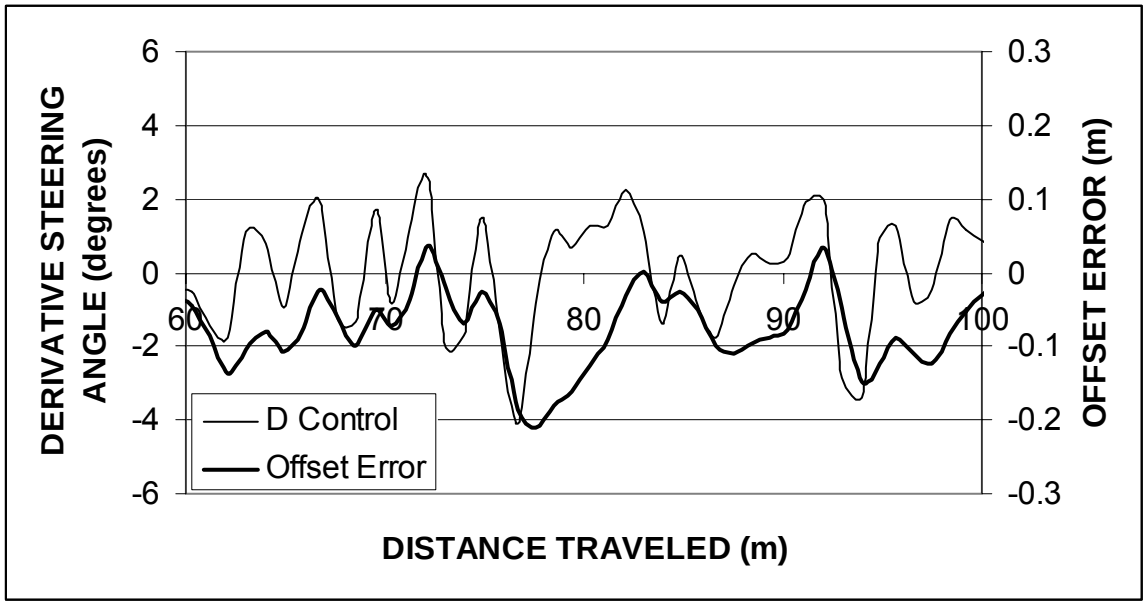


Figure 38. Derivative Gain Impact on Steady State Error

ii. Position-Based Digital PID Controller

A simpler digital PID model was developed to reduce the computational load of the task computer, improve the error estimation, and incorporate gain scheduling. Digital implementation of a PID controller can be derived in many forms. Multiple forms were considered and analyzed based on their relevance to this application method. The chosen model was based on a position PID controller, which transforms the analog PID model into a digital form by using rectangular integration approximation. The general form of the controller accounted for the current position error as well as the previous two position errors. An auxiliary term was added to the control algorithm to account for the center position of the steering actuator. The sampling time was held constant at 2 Hz and was lumped into the values for the gains.

$$C_{PID} = CP + [P_g \times E_K] + [I_g \times (E_K + E_{K-1} + E_{K-2})] + [D_g \times (E_K - E_{K-1})] \quad (13)$$

Where:

C_{PID} = Current Control Signal

CP = Center Location of Steering Actuator

E_K = Current Average Error (cm)

E_{K-1} = Previous Average Error (cm)

E_{K-2} = Second Previous Average Error (cm)

As equation 13 shows, when all error values were driven to zero, the output will simply be the calibrated value for the center location of the steering actuator, thus the vehicle will travel straight. It was noticed in the previous controller that small random errors in the GPS signal caused significant changes in the control signal relative to the magnitude of the error. To counteract this phenomenon, three consecutive error values were averaged together and used as the current average error value. The two previous average error values were calculated in a similar manner, thus five data points were used to determine each steering control command. This served as a filter for the GPS points and dramatically smoothed the offset error values.

Gain scheduling was incorporated to reduce the effects of large gain values when the offset error was small, thus eliminating the impulsive motions seen in the previous controller design. Previous testing data were used to determine the scheduling parameters. The derivative gain value was set to zero whenever the value of $(E_K - E_{K-1})$ became less than 0.5 cm. Likewise, the integral gain was not activated in the system unless the value of E_K was less than 15 cm.

This algorithm was used to guide the vehicle through a standard step response, which was identical to the step response used to test the previous controller. The P & D gain variables were tuned to stabilize the vehicle response. It was found that a proportional gain value of 0.75 and a derivative gain value of 7 produced a damping ratio of 0.65. There was still a steady state error of 0.11 m apparent in the step response curve. Several values of integral gains were tested to reduce the amount of steady state offset error (Figure 39).

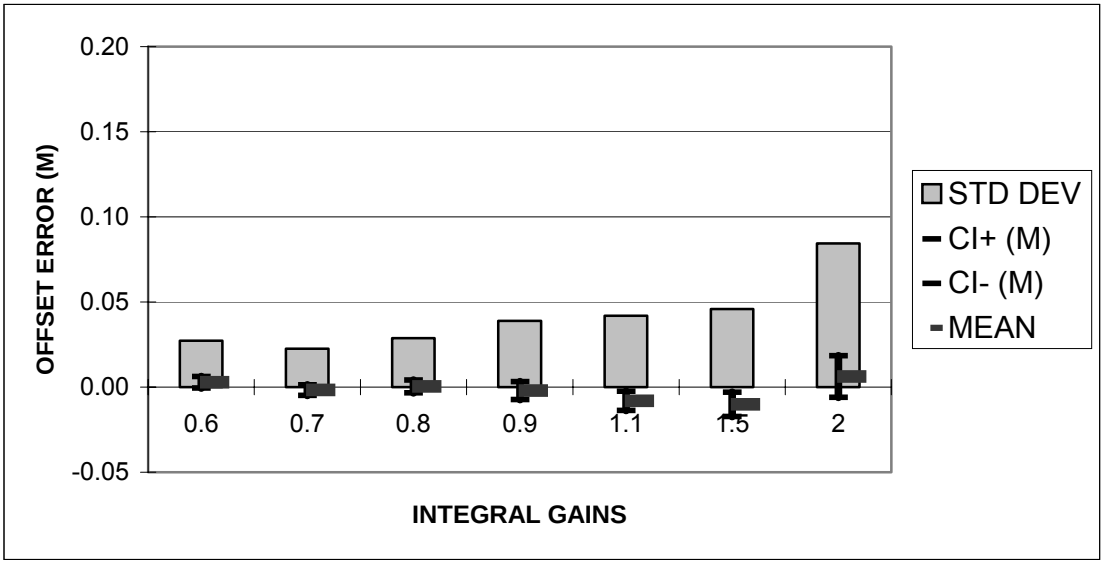


Figure 39. Version 2 Controller Step Response Data for Various Integral Gains

As seen in previous testing, the standard deviation of steady state errors increased with an increase in integral gain (Figure 39). An integral gain of 0.8 was chosen for this application. This provided a mean offset error of 0.34 mm and a standard deviation of 28.7 mm. The Version 2 controller significantly improved the step response of the autonomous vehicle by reducing the computation load of the task computer, which in turn decreased the time delay between reception of a new position point and transmission of a control command. Incorporating error filtering and gain scheduling routines improved the standard deviation of the guidance data by reducing the amount of impulsive control commands.

iii. Control System Validation Testing at Increased Ground Speed

All previous tests were conducted at a ground speed of 0.83 m/s. Further testing using the Version 2 PID controller design was conducted at a forward ground speed of 1.7 m/s. Integral gains from 0.6 to 0.9 were used to evaluate any differences in the system response caused by a higher ground speed. The proportional gain was held at 0.75 and the derivative gain at 7 for all tests.

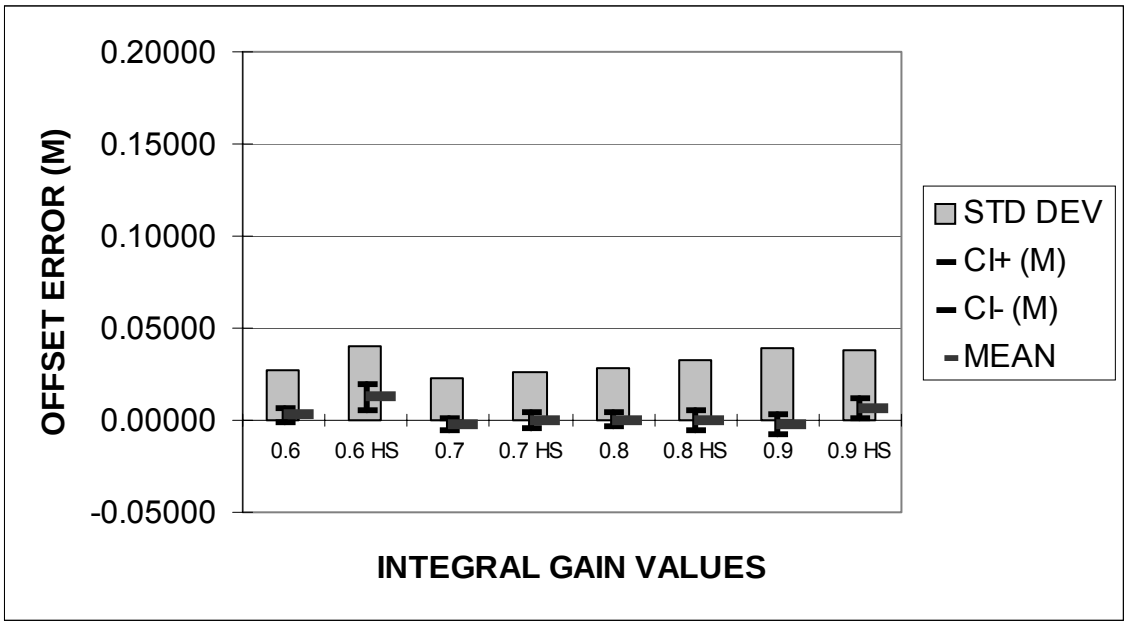


Figure 40. Version 2 Controller Step Response Data for Various Integral Gains and High Ground Speed

Analyses showed that at a higher ground speed of 1.7 meters per second, an integral gain of 0.8 was still optimal (Figure 40). This produced a steady state offset error of 0.05 mm and a standard deviation of 32.3 mm. These error values were well within the design limits of the project and showed that the control system could be applied to a faster autonomous vehicle. Unfortunately, 1.7 m/s was the maximum forward speed of the test vehicle; thus, the kinematic model threshold of 3.3 m/s could not be tested.

One anomaly was revealed during the increased speed testing. An increase in the forward vehicle speed should have resulted in a decrease in system damping and thus a larger overshoot and smaller damping ratio. Results from the increased speed testing showed an overshoot of 3.6% and a damping ratio of 0.725. The result was justified by reevaluating the governing control equation. Because the vehicle was traveling faster during the high speed tests, the difference between the current point and previous point had a greater magnitude than in the previous tests. This increased magnitude allowed the derivative component to play a larger role in determining the response of the system and resulted in a system with increased dampening.

D. Demonstrate the Ability to Traverse a Normal Field Operation

The final objective of the project was to demonstrate the autonomous capabilities of the vehicle by completing consecutive headland turning and straight-line guidance routines. Figure 41 represents the path used to test the vehicle's autonomous capabilities. This path was often seen in production agricultural operations and demonstrated the feasibility of the vehicle for this application. Only the initial two guidance points and the swath width between the operating passes were supplied to the vehicle.

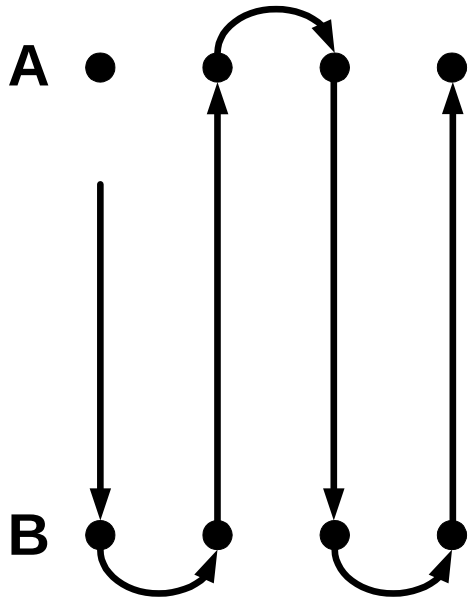


Figure 41. Field Operation Schematic

Several objectives were set for the vehicle to effectively traverse the field operation.

1. Develop an algorithm to determine when the headland area had been reached
2. Develop an algorithm to calculate the next guidance line based on the current guidance line and defined vehicle swath width
3. Utilize a digital compass to implement an open loop headland turning routine.

A simple algorithm was developed to determine when the headland area had been reached thus requiring initiation of a turning routine. The first step in the algorithm was to determine whether the vehicle was heading towards point A or point B. Then, the distance from the current location to the origination point and the distance between the two initial set-points were compared (Figure 42). If the vehicle had passed into the headland area, then the distance between its current location and the point which originated the line of travel was greater than the distance between the two points used to define the line of travel. Full implementation of this routine can be seen in Appendix B.

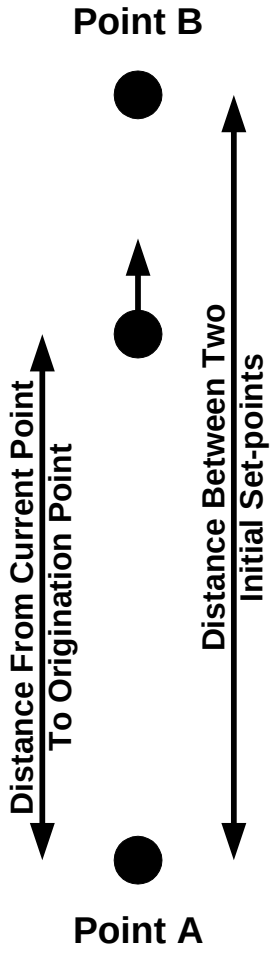


Figure 42. Headland Logic

The next step in completing a field operation required that the task computer be able to calculate the next desired guidance path. Two points, A and B, defined the

initial path. When the vehicle extended beyond either of these end points as discussed above, the control system redefined points A and B based on the set swath width of the field operation (Equations 14, 15, 16, & 17). This newly defined line was parallel to the initial line and contained new A-B points that were perpendicular to the initial points (Figure 43).

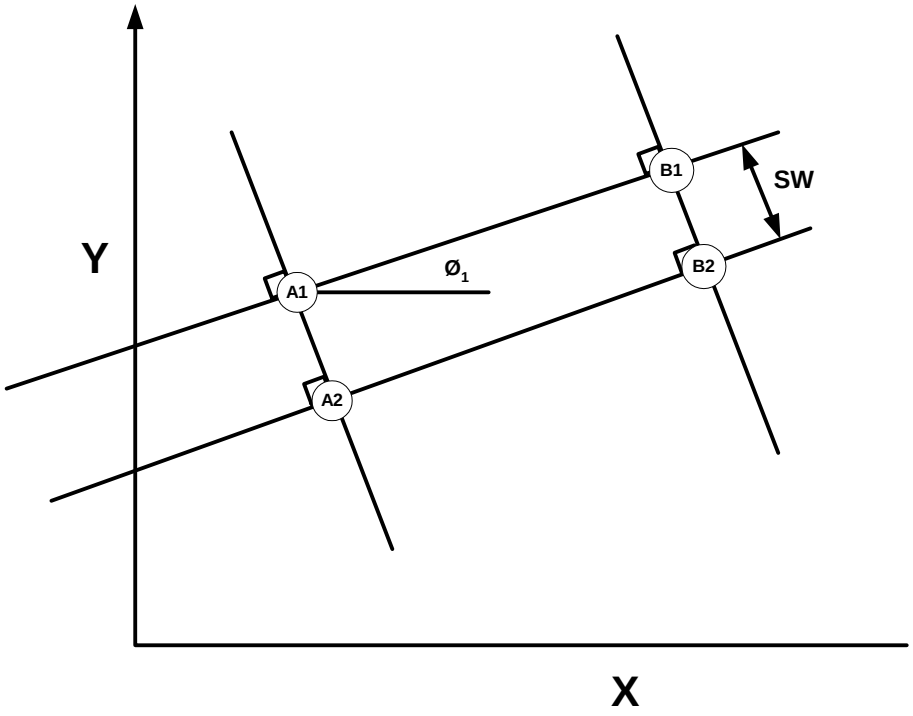


Figure 43. Guidance Line Translation

Points A2 and B2 were calculated based on the following equations:

$$B_{2x} = \left[\frac{B_{1y} + \frac{1}{m_1} B_{1x} - A_{1y} + m_1 A_{1x} + \frac{SW}{\cos\theta_1}}{m_1 + \frac{1}{m_1}} \right] \quad (14)$$

$$B_{2y} = \left[m_1 B_{2x} + A_{1y} - m_1 A_{1x} - \frac{SW}{\cos\theta_1} \right] \quad (15)$$

$$A_{2x} = \left[\frac{A_{1y} + \frac{1}{m_1} A_{1x} - A_{1y} + m_1 A_{1x} + \frac{SW}{\cos\theta_1}}{m_1 + \frac{1}{m_1}} \right] \quad (16)$$

$$A_{2y} = \left[m_1 A_{2x} + A_{1y} - m_1 A_{1x} - \frac{SW}{\cos\theta_1} \right] \quad (17)$$

Where:

A_{1x} = Initial Point A Easting

A_{1y} = Initial Point A Northing

B_{1x} = Initial Point B Easting

B_{1y} = Initial Point B Northing

m_1 = Initial Slope of Line AB

SW = Swath Width

$\theta_1 = \tan^{-1}(m_1)$

Open loop control was used during specialized routines such as headland turning operations. When the end of the row was reached during a normal field operation the vehicle was required to lift the rear implement, turn around, and beginning tracking the next guidance line. To accomplish these functions the task computer sent a single message specifying that the end of the row had been reached and specifying the direction and distance to the next guidance line. The steering ECU received this message and took appropriate action to steer the vehicle toward the next line. The vehicle maintained a constant steering angle while maneuvering the turn. The steering ECU monitored a digital compass and determined when the vehicle had changed direction by 180 ±15 degrees, thus marking the end of the headland turning routine. The steering ECU then notified all nodes that the headland turning operation had ended

by transmitting a single CAN message. The source code for this operation is included in Appendix C.

During the time it took the vehicle to maneuver the turn, the task computer was able to calculate the new guidance line and prepared to resume autonomous guidance. The transmission node responded to the start and finish of the end of the row operation message by slowing the vehicle speed during the turning maneuver. The speed setting was adjusted using the configure transmission message (Appendix A). Also, the hitch node recognized the start and finish of the headland turning message and raised or lowered the implement appropriately.

The efficiency of this turning routine was based on the accuracy of predicting the appropriate steering angle. Initial testing was completed by using the bicycle model to predict the steering angle based on the turning radius, which was one half of the implement swath width. The bicycle model defined the turning rate of a vehicle as the forward vehicle velocity divided by the radius of turn or half the diameter of turn (Equation 18). For this vehicle, the forward velocity was 0.83 meters per second and the working swath width was between 7 and 20 meters. The working range of the vehicle turning rate was then 6.79 to 2.37 °/sec.

$$\frac{d\phi}{dt} = \frac{V}{R1} \qquad (18)$$

Where: V = forward velocity of the vehicle

$R1$ = effective turning radius of the vehicle

This control system was tested in a field operation simulation. Two initial set-points and a swath width were loaded into the task computer. The vehicle autonomously tracked the straight line between the points. It then determined when it had passed the farthest assigned point and began the standardized turning routine based on a given swath width. The vehicle successfully traversed the field and demonstrated appropriate turning functionality (Figure 44). It was found that while the deadband of ±15° worked well, it could have been reduced. During every recorded turning event, the vehicle released from the turning routine on the lower side of the deadband, thus before a full 180° turn had been made.

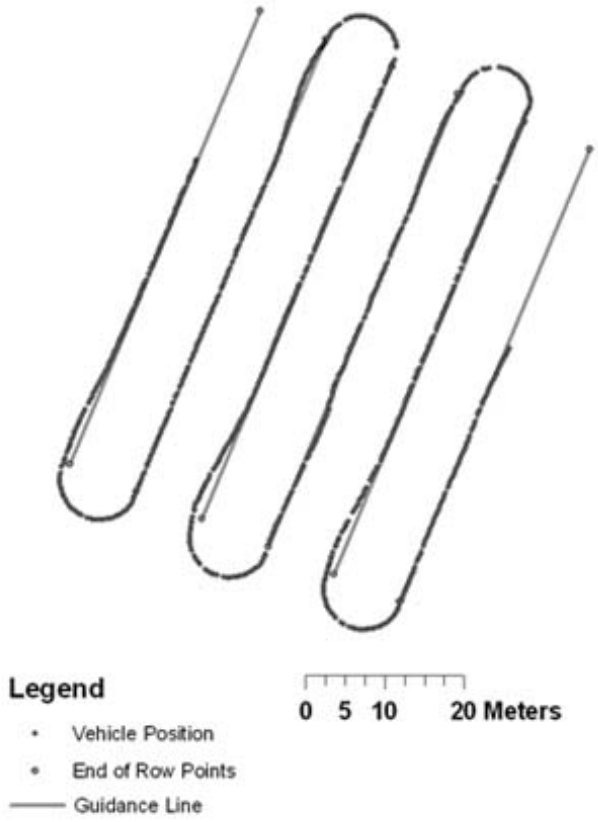


Figure 44. Field Path Demonstration

E. System Cost

The overall system cost was low relative to alternative designs for autonomous vehicle control. Utilizing inexpensive microcontrollers at each node greatly reduced the data processing expense.

Table 8. Cost of Individual ECU and CAN Interface

Component	Units Required	Unit Price	Total Cost
PIC 18F258	1	$ 6.17	$ 6.17
Amp Connectors	6	$ 5.75	$ 34.50
28-pin Socket	1	$ 2.26	$ 2.26
Printed Circuit Board	1	$ 2.00	$ 2.00
Screw Terminal	2	$ 1.75	$ 3.50
Enclosure	1	$ 1.75	$ 1.75
MCP 2551	1	$ 1.48	$ 1.48
2A Fuse	1	$ 0.84	$ 0.84
20 MHz Clock	1	$ 0.75	$ 0.75
8-Pin Socket	1	$ 0.60	$ 0.60
LM7805 - 1A Reg	1	$ 0.50	$ 0.50
Switch	1	$ 0.43	$ 0.43
LM78L05 - .1A Reg	1	$ 0.40	$ 0.40
Diode	1	$ 0.27	$ 0.27
Connector Pins	54	$ 0.26	$ 14.04
22pF Cap	2	$ 0.14	$ 0.28
0.1µF Cap	1	$ 0.08	$ 0.08
Total			**$ 69.84**

The overall unit cost per node was $69.84 (Table 8). This included all components required to correctly operate the microcontroller and transceiver chips as well as all materials required to enclose the ECU. The connector cost included two CAN connections per box as well as a third connection for an auxiliary sensor. The cost of the female connectors that attached to the ECU male connectors were also included in the ECU analysis rather than developing a separate cost summary. Seventy percent of the overall cost of the ECU was directly related to the connectors. Practically this is a high percentage to allocate to the physical connection between the node and the bus, but it was critical to use quality connections to ensure the reliability of the bus.

Table 9. Cost of System Wide Sensors

Component	Units Required	Unit Price	Total Cost
Sub-meter GPS Receiver	1	$3,000.00	$3,000.00
Task Computer	1	$1,200.00	$1,200.00
Steering Actuator	1	$ 750.00	$ 750.00
Transmission Actuator	1	$ 150.00	$ 150.00
Hitch Actuator	1	$ 150.00	$ 150.00
High Current Motor Controller	1	$ 85.00	$ 85.00
Radio Frequency Controller	1	$ 80.00	$ 80.00
Digital Compass	1	$ 50.00	$ 50.00
Piezo-electric Rate Sensor	1	$ 25.00	$ 25.00
IC H-Bridge	2	$ 13.00	$ 26.00
Total			**$5,516.00**

The total amount allocated to sensors was $5,516.00 (Table 9). This cost could fluctuate substantially depending on the type of GPS receiver used. Also, a single board computer could be used rather than a laptop task computer to reduce cost.

Chapter 5: Conclusions

A controller area network provided an efficient platform to develop an autonomous vehicle control system. Individual control nodes reduced the computational load of the task computer by implementing the feedback control logic at the node. Additional nodes were easily added to the system to provide increased functionality or feedback. The CAN-RS232 bridge was an inexpensive means to transfer CAN based data into a task computer for data logging and mathematical computation of autonomous control commands. A digital PID controller was sufficient in controlling the steering system with a high degree of accuracy. Gain scheduling and GPS position filtering reduced the occurrence of impulsive commands during situations where the guidance errors were very low. When operated within the parameters of the kinematic model, the control system exhibited errors that were nearly negligible. A low cost piezo-electric turning rate sensor was not able to correctly determine the turning rate of the vehicle due to the high level of mechanical vibrations from the host vehicle.

An open loop headland turning routine was successfully implemented using a digital two-dimensional compass as the means of directional feedback.

Chapter 6: Future Work

Several future research topics were encountered during the completion of this project. The transceiver chips were a problem throughout the course of the project, most likely due to improper design of the printed circuit board. More work is required to develop a more reliable circuit board. Also, the task computer program should be migrated from the laptop computer to a hardened single board computer. This will reduce the latency caused by the operating system environment. Fault protection should also be incorporated into the task computer algorithm.

Vehicle guidance should be improved by developing a self-calibration routine that the steering ECU performs during initialization. This could be as simple as mounting a limit switch that would relate to the center point of the steering axle. The guidance routine could be improved by incorporating a direction command with the predefined AB guidance line. This would provide the control system with an initial target heading and reduce errors at the onset of a guidance operation. A turning rate sensor with higher sensitivity could improve the headland turn routine by providing an accurate feedback signal. Incorporating this sensor would create an entirely closed loop control machine, which would improve reliability. Finally, any autonomous machine should have a means of obstacle detection and collision avoidance. This vehicle should have a basic mechanism to sense its immediate surroundings and prevent any potentially disastrous situations.

Appendices

Appendix A: CAN Node Commands

Steering Command	
Description	Desired Steering Angle
Source Address	1
Priority	3
Transmission Frequency (Hertz)	5
Data Byte 0	Least Significant Byte of Desired Steering Position in Feedback Counts
Data Byte 1	Most Significant Byte of Desired Steering Position in Feedback Counts

Transmission Command	
Description	Desired Transmission Position
Source Address	2
Priority	3
Transmission Frequency (Hertz)	5
Data Byte 0	Hydrostatic Transmission Position in Feedback Counts

Hitch Command	
Description	Desired Hitch Position
Source Address	3
Priority	3
Transmission Frequency (Hertz)	5
Data Byte 0	Hitch Position in Feedback Counts

Start Turning Routine	
Description	Notify Nodes to Begin Turning Routine (end of row operation)
Source Address	8
Priority	5
Transmission Frequency (Hertz)	
Data Byte 0	Least Significant Byte of Desired Steering Position in Feedback Counts
Data Byte 1	Most Significant Byte of Desired Steering Position in Feedback Counts
Data Byte 2	Set as 1 to turn. All other values will do nothing.

Stop Operation	
Description	Stop Vehicle Movement, Center Steering Axle, Raise Implement
Source Address	12
Priority	0
Transmission Frequency (Hertz)	

Configure Steering Node

Description	Configure Parameters within the Steering Node
Source Address	10
Priority	5
Transmission Frequency (Hertz)	
Data Byte 0	New Deadband Value
Data Byte 1	Not Used
Data Byte 2	New Lower Turn Angle Limit
Data Byte 3	New Upper Turn Angle Limit

Configure Transmission Node

Description	Configure Parameters within the Transmissin Node
Source Address	11
Priority	5
Transmission Frequency (Hertz)	
Data Byte 0	New Deadband Value
Data Byte 1	Not Used
Data Byte 2	New Relay Ontime Value. Not Used in Latest Software with H-Bridge Function

Configure Hitch Node

Description	Configure Parameters within the Hitch Node
Source Address	13
Priority	5
Transmission Frequency (Hertz)	

Configure GPS Node

Description	Configure Parameters within the GPS Node
Source Address	14
Priority	5
Transmission Frequency (Hertz)	

Steering Turning Routine Complete

Description	Steering has Returned to Normal Operation
Source Address	9
Priority	5
Transmission Frequency (Hertz)	
Data Byte 0	Equal to 1 for Completed Turning Routine.

Transmission Turning Routine Complete

Description	Transmission has Returned to Normal Operation
Source Address	15
Priority	5
Transmission Frequency (Hertz)	

Hitch Turning Routine Complete

Description	Hitch has Returned to Normal Operation
Source Address	16
Priority	5
Transmission Frequency (Hertz)	

GPS Position

Description	Vehicle Positition based on GPS Receiver
Source Address	5
Priority	2
Transmission Frequency (Hertz)	5
Data Byte 0	Latitude Byte 0
Data Byte 1	Latitude Byte 1
Data Byte 2	Latitude Byte 2
Data Byte 3	Latitude Byte 3
Data Byte 4	Longitude Byte 0
Data Byte 5	Longitude Byte 1
Data Byte 6	Longitude Byte 2
Data Byte 7	Longitude Byte 3

Time

Description	Time based on GPS Receiver
Source Address	6
Priority	2
Transmission Frequency (Hertz)	5
Data Byte 0	Hours
Data Byte 1	Minutes
Data Byte 2	Seconds

GPS Quality

Description	GPS Quality Information
Source Address	7
Priority	2
Transmission Frequency (Hertz)	5
Data Byte 0	Differential Correction Status
Data Byte 1	Number of Satellites

GPS Message Processing

Incoming Latitude Format: ddmm.nnnooo

Incoming Longitude Format: eepp.qqqrrr

Variable Name	Value
LATDD	dd
LATMM	mm
LATMMM1	nnn
LATMMM2	ooo
LONDD	ee
LONMM	pp
LONMMM1	qqq
LONMMM2	rrr

Outgoing GPS Message Format:

Data Byte	Value
TXnD0	Low Byte of (LATDD * 1000) + LATMMM1
TXnD1	High Byte of (LATDD * 1000) + LATMMM1
TXnD2	Low Byte of (LATMM * 1000) + LATMMM2
TXnD3	High Byte of (LATMM * 1000) + LATMMM2
TXnD4	Low Byte of ((LONDD - 40) * 1000) + LONMMM1
TXnD5	High Byte of ((LONDD - 40) * 1000) + LONMMM1
TXnD6	Low Byte of (LONMM * 1000) + LONMMM2
TXnD7	High Byte of (LONMM * 1000) + LONMMM2

Appendix B: Task Computer Program Code

```
Const PI As Single = 3.1415927
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
Option Explicit
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub ACCEPT_BUTTON_Click()
'WHEN THE ACCEPT BUTTON IS PRESSED THE A AND B POINTS ARE USED TO
'CALCULATE STEERING PARAMETERS

X1 = POINT_A_EAST_TEXT.Text
Y1 = POINT_A_NORTH_TEXT.Text
X2 = POINT_B_EAST_TEXT.Text
Y2 = POINT_B_NORTH_TEXT.Text

A = (Y2 - Y1) / (X2 - X1)
B = (-1)
C = Y1 - (A * X1)

UKAT_AUTOSTEER.Enabled = True

End Sub
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub CALC_LINE_Click()
'WHEN THE CALC_LINE BUTTON IS PRESSED A NEW GUIDANCE LINE IS CALCULATED BASED ON
'THE CURRENT GUIDANCE LINE AND THE DESIRED SWATH WIDTH
   M1 = A
   B1 = C
   TURNING_COEF = TURNING_DIR_TEXT.Text    'THIS COEFFICIENT DESIGNATES RIGHT OR LEFT TURN
   THETA1 = Atn(M1)
   M2 = M1
   DB = (TURNING_COEF * SWATH_WIDTH) / Cos(THETA1)
   B2 = (B1 - DB)
   M3 = -1 / M1
   B3 = Y2 - (M3 * X2)
   M4 = M3
   B4 = Y1 - (M4 * X1)
   X2 = (B3 - B2) / (M2 - M3)
   Y2 = (M2 * X2) + B2
   X1 = (B4 - B2) / (M2 - M4)
   Y1 = (M2 * X1) + B2
   A = (Y2 - Y1) / (X2 - X1)
   B = (-1)
   C = Y1 - (A * X1)
   CALC_LINE.Value = False
End Sub
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub CANCEL_BUTTON_Click()
'THE CANCEL BUTTON CLEARS ALL A AND B VALUES
   SET_B_BUTTON.Enabled = False
   ACCEPT_BUTTON.Enabled = False
   SET_A_BUTTON.Enabled = True
   UKAT_AUTOSTEER.Enabled = False
   POINT_A_EAST_TEXT.Text = " "
   POINT_B_EAST_TEXT.Text = " "
   POINT_A_NORTH_TEXT.Text = " "
   POINT_B_NORTH_TEXT.Text = " "

End Sub
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub LOWER_HITCH_BUTTON_Click()
'THE LOWER_HITCH_BUTTON MANUALLY LOWERS THE THREE POINT HITCH

            While TIMEOUT < 5000
               TIMEOUT = TIMEOUT + 1
            Wend
            TIMEOUT = 0
            MSComm1.Output = "$3,200,0,0,0,0,0,0,0,0,0#"
            While TIMEOUT < 5000
```

```
            TIMEOUT = TIMEOUT + 1
        Wend
        TIMEOUT = 0
        DoEvents
End Sub
---------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub EXIT_BUTTON_Click()
'CLOSE PORTS AND EXIT PROGRAM

    ProLatUTMClose (iHandle)
    MSComm1.PortOpen = False
    Close
    End
End Sub
---------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub FILENAME_BUTTON_Click()
'FILENAME_BUTTON ALLOWS USER TO NAME FILE TO STORE DATA

    On Error Resume Next

    CommonDialog1.DialogTitle = "CHOOSE DATA FILENAME"
    CommonDialog1.ShowOpen

    'FILENAME_TEXT.Text = CommonDialog1.FileName

    Open CommonDialog1.FileName For Append As #1

    Print #1, "X1, Y1, X2, Y2, GPS_TIME, GPS_QUALITY, NUMSAT, ESTNG, NRTNG, ESTNG_PREV, NRTNG_PREV, VAR1,
D_GAIN, P_GAIN, I_GAIN, AB, AC, BC, AC_PREV, BC_PREV, UKAT_AUTOSTEER.Value, UKAT_AUTOTURN.Value, E_0, E_1,
E_2, E_K, E_K_1, E_K_2, C_PID, CENTER_POINT, HEADING_DIFF"
    'THE PRINT LINE ADDS HEADERS TO THE DATA TABLE

    OPENFILE = True

End Sub
---------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub Form_Load()
'INTIALIZE PROGRAM

    'SET UP COMM PORT
    MSComm1.CommPort = 5
    MSComm1.Settings = "19200,n,8,1"
    ROUNDUP_HEADING = False

    If MSComm1.PortOpen = True Then
        MsgBox "THE PORT IS ALREADY IN USE"
        Exit Sub
    End If

    MSComm1.PortOpen = True

    'INITIALIZE UTM CONVERTER .dll
    iHandle = ProLatUTMInitialize("WGS84", 0, 1)

    CommonDialog1.InitDir = "C:\DARR\UKAT PROGRAMS"

    roundup.Show

End Sub
---------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub MSComm1_OnComm()
'MSComm1 ROUTINE DEFINES OPERATIONS ON NEW SERIAL MESSAGE RECEIVE INTERRUPT

    Const PI As Single = 3.1415927
    On Error Resume Next

    If MSComm1.CommEvent = comEvReceive Then    'CHECK FOR NEW MESSAGE RECEIVED
        BUFFER_LENGTH = MSComm1.InBufferCount
    Else
        BUFFER_LENGTH = 0
```

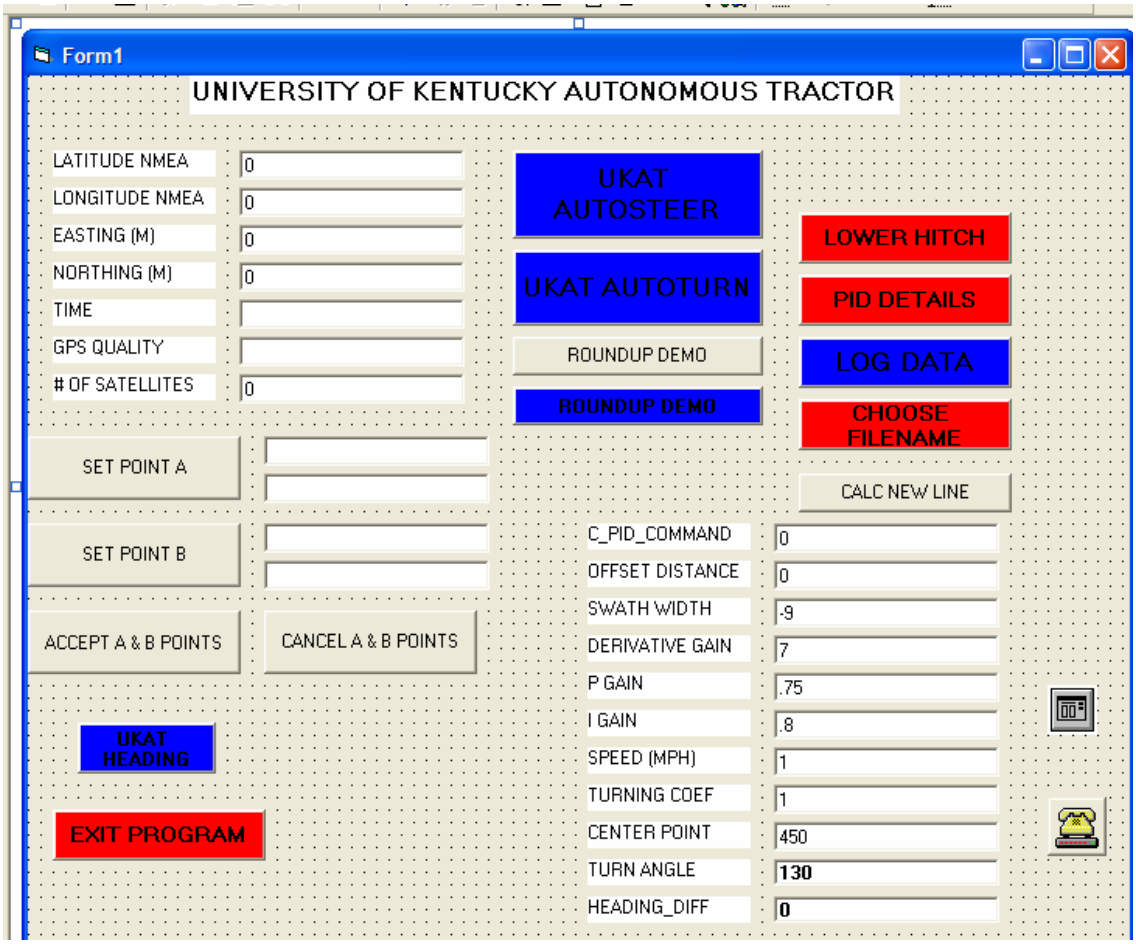
```
End If

While BUFFER_LENGTH > 5
   BUFFER_ARRAY = BUFFER_LEFTOVER & MSComm1.Input  'ADD NEW MESSAGE TO BUFFER
   BUFFER_LENGTH = Len(BUFFER_ARRAY)
   START_POS = InStr(BUFFER_ARRAY, "$")    'DEFINE START OF MESSAGE STRING
   END_POS = InStr(BUFFER_ARRAY, "#")      'DEFINE END OF MESSAGE STRING

   If START_POS = 0 Then Exit Sub
   If END_POS = 0 Then Exit Sub

   DATA_STRING = Mid(BUFFER_ARRAY, START_POS + 1, END_POS - START_POS - 1) 'SPLIT DATA STRING
   DATA_ARRAY = Split(DATA_STRING, ",")
   SA = DATA_ARRAY(0)                    'DEFINE SOURCE ADDRESS OF NEW MESSAGE
   BUFFER_LEFTOVER = Mid(BUFFER_ARRAY, END_POS + 1)    'COLLECT UNUSED BUFFER

   If SA = 5 Then                        'NEW GPS POINT
      ESTNG_PREV = ESTNG                 'SAVE PREVIOUS POINTS
      NRTNG_PREV = NRTNG
      SA = 0                          'RESET SA
      LATDD = DATA_ARRAY(1)              'ASSIGN ARRAY STRINGS
      LATMM = DATA_ARRAY(2)
      LATMMM1 = DATA_ARRAY(3)
      LATMMM2 = DATA_ARRAY(4)
      LONDD = DATA_ARRAY(5)
      LONMM = DATA_ARRAY(6)
      LONMMM1 = DATA_ARRAY(7)
      LONMMM2 = DATA_ARRAY(8)
      LAT_MINUTE = ((LATMM * (10 ^ 6)) + (LATMMM1 * (10 ^ 3)) + LATMMM2) / (10 ^ 6)
      LAT_DEC_DEGREE = LAT_MINUTE / 60
      LAT = LATDD + LAT_DEC_DEGREE
      LON_MINUTE = ((LONMM * (10 ^ 6)) + (LONMMM1 * (10 ^ 3)) + LONMMM2) / (10 ^ 6)
      LON_DEC_DEGREE = LON_MINUTE / 60
      LON = (LONDD + LON_DEC_DEGREE) * -1
      LAT_NMEA_TEXT.Text = LAT
      LON_NMEA_TEXT.Text = LON
      ICODE = ProLatUTMTransform(iHandle, LON, LAT, ESTNG, NRTNG, 0)  'CONVERT GPS POINTS TO UTM
      EASTING_TEXT.Text = ESTNG
      NORTHING_TEXT.Text = NRTNG
      DELTA_X = (ESTNG - ESTNG_PREV)
      DELTA_Y = (NRTNG - NRTNG_PREV)

      'DETERMINE WHICH CARTESIAN QUADRANT THE SYSTEM IS IN
      If DELTA_X > 0 And DELTA_Y > 0 Then HEADING_DIR = Atn(DELTA_Y / DELTA_X) * 180 / PI 'QUAD1
      If DELTA_X < 0 And DELTA_Y > 0 Then HEADING_DIR = 180 + (Atn(DELTA_Y / DELTA_X) * 180 / PI) 'QUAD2
      If DELTA_X < 0 And DELTA_Y < 0 Then HEADING_DIR = 180 + (Atn(DELTA_Y / DELTA_X) * 180 / PI) 'QUAD3
      If DELTA_X > 0 And DELTA_Y < 0 Then HEADING_DIR = 360 + (Atn(DELTA_Y / DELTA_X) * 180 / PI) 'QUAD4
      C1 = NRTNG - (ESTNG * A)
      HEADING = ESTNG - ESTNG_PREV

      'CONTROL VARIABLE
      If C > C1 Then
         VAR1 = -1
      Else
         VAR1 = 1
      End If

      'SET SIGN CONVENTION
      If HEADING > 0 Then HEADING_FACTOR = 1
      If HEADING < 0 Then HEADING_FACTOR = -1
      If HEADING = 0 Then HEADING_FACTOR = 0

      'CALCULATE CONTROL ERRORS AND SETPOINT PARAMETERS
      D = Abs(A * ESTNG + B * NRTNG + C) / ((A ^ 2 + B ^ 2) ^ (0.5))
      OFFSET_DIST_TEXT.Text = D * VAR1
      D_GAIN = D_GAIN_TEXT.Text * HEADING_FACTOR
      P_GAIN = P_GAIN_TEXT.Text * HEADING_FACTOR
      I_GAIN = I_GAIN_TEXT.Text * HEADING_FACTOR

      'APPLY GAIN SCHEDULING
```

```
If Abs(E_K) > 15 Then I_GAIN = 0
If Abs(E_K - E_K_1) < 0.5 Then D_GAIN = 0

'CALCULATE PARAMETERS FOR HEADLAND TURN DECISION TABLE
AB = Sqr((X2 - X1) ^ 2 + (Y2 - Y1) ^ 2)
AC = Sqr((X1 - ESTNG) ^ 2 + (Y1 - NRTNG) ^ 2)
AC_PREV = Sqr((X1 - ESTNG_PREV) ^ 2 + (Y1 - NRTNG_PREV) ^ 2)
BC = Sqr((X2 - ESTNG) ^ 2 + (Y2 - NRTNG) ^ 2)
BC_PREV = Sqr((X2 - ESTNG_PREV) ^ 2 + (Y2 - NRTNG_PREV) ^ 2)

SWATH_WIDTH = SWATH_WIDTH_TEXT.Text
M1 = A
B1 = C
CENTER_POINT = CENTER_POINT_TEXT.Text   'STEERING ACTUATOR CALIBRATED CENTER

'AVERAGE THREE PREVIOUS ERROR VALUES AND RECORD PREVIOUS ERRORS
E_2 = E_1
E_1 = E_0
E_0 = (D * VAR1) * 100
E_K_2 = E_K_1
E_K_1 = E_K
E_K = (E_0 + E_1 + E_2) / 3

'APPLY PID CONTROLLER
C_PID = CENTER_POINT + P_GAIN * E_K + I_GAIN * (E_K + E_K_1 + E_K_2) + D_GAIN * (E_K - E_K_1)

'SET MAX AND MIN LIMITS ON PID CONTROL COMMAND
If C_PID < 100 Then C_PID = 100
If C_PID > 840 Then C_PID = 840

'NORMAL AUTOGUIDANCE MODE
If UKAT_AUTOSTEER.Value = 1 And ROUNDUP_HEADING = False Then

   'OUTPUT PID CONTROL COMMAND
   C_PID_TEXT.Text = C_PID
   MSComm1.Output = "$1," & C_PID & ",0,0,0,0,0,0,0,0,0#"

   'ALLOW CAN BRIDGE TIME TO TRANSMIT MESSAGE
   While TIMEOUT < 5000
      TIMEOUT = TIMEOUT + 1
   Wend
   TIMEOUT = 0

   'OUTPUT TRANSMISSION SPEED COMMAND
   MSComm1.Output = "$2,220,0,0,0,0,0,0,0,0,0#"
   While TIMEOUT < 5000
      TIMEOUT = TIMEOUT + 1
   Wend
   TIMEOUT = 0

   'OUTPUT THREE POINT HITCH COMMAND
   MSComm1.Output = "$3,200,0,0,0,0,0,0,0,0,0#"
   While TIMEOUT < 5000
      TIMEOUT = TIMEOUT + 1
   Wend
   TIMEOUT = 0
   DoEvents
End If

'DETERMINE IF HEADLAND TURN IS REQUIRED
If UKAT_AUTOTURN.Value = 1 And ROUNDUP_HEADING = False Then
   If AC > AB And AC > AC_PREV Then    'TURN RIGHT
      TURN_ANGLE = CENTER_POINT + 330
      MSComm1.Output = "$1," & TURN_ANGLE & ",0,0,0,0,0,0,0,0,0#"
      While TIMEOUT < 5000
         TIMEOUT = TIMEOUT + 1
      Wend
      ROUNDUP_HEADING = True
      INITIAL_HEADING = HEADING_DIR
      CALC_LINE.Value = True
```

```
            DoEvents
        End If
        If BC > AB And BC > BC_PREV Then    'TURN LEFT
            TURN_ANGLE = CENTER_POINT - 320
            MSComm1.Output = "$1," & TURN_ANGLE & ",0,0,0,0,0,0,0,0#"
            While TIMEOUT < 5000
                TIMEOUT = TIMEOUT + 1
            Wend
            ROUNDUP_HEADING = True
            INITIAL_HEADING = HEADING_DIR
            CALC_LINE.Value = True
            DoEvents
        End If
    End If

    'TRAVERSE HEADLAND TURN
    If ROUNDUP_HEADING = True Then

        'COLLECT HEADING INFORMATION
        HEADING_DIFF = Abs(INITIAL_HEADING - HEADING_DIR)
        HEADING_TEXT.Text = HEADING_DIFF

        'TRANSMIT CONSTANT TURNING ANGLE
        MSComm1.Output = "$1," & TURN_ANGLE & ",0,0,0,0,0,0,0,0#"
        While TIMEOUT < 5000
            TIMEOUT = TIMEOUT + 1
        Wend
        TIMEOUT = 0

        'TRANSMIT CONSTANT TRANSMISSION SPEED
        MSComm1.Output = "$2,200,0,0,0,0,0,0,0,0#"
        While TIMEOUT < 5000
            TIMEOUT = TIMEOUT + 1
        Wend
        TIMEOUT = 0

        'TRANSMIT RAISED IMPLEMENT HEIGHT
        MSComm1.Output = "$3,50,0,0,0,0,0,0,0,0#"
        While TIMEOUT < 5000
            TIMEOUT = TIMEOUT + 1
        Wend
        TIMEOUT = 0

        'DETERMINE IF TURN IS COMPLETE
        If HEADING_DIFF > 165 And HEADING_DIFF < 195 Then
            MSComm1.Output = "$1," & CENTER_POINT & ",0,0,0,0,0,0,0,0#"
            While TIMEOUT < 5000
                TIMEOUT = TIMEOUT + 1
            Wend
            ROUNDUP_HEADING = False
        End If
    End If

    'LOG DATA IF REQUESTED
    If LOG_DATA.Value = 1 Then
        'Warn the user if no file name was selected
        If Not OPENFILE Then
            LOG_DATA.Value = 0
            MsgBox ("Select filename before logging.")
            Exit Sub
        End If
        CPU_SEC = Second(Time)
        Write #1, X1, Y1, X2, Y2, GPS_TIME, GPS_QUALITY, NUMSAT, ESTNG, NRTNG, ESTNG_PREV, NRTNG_PREV,
VAR1, D_GAIN, P_GAIN, I_GAIN, AB, AC, BC, AC_PREV, BC_PREV, UKAT_AUTOSTEER.Value, UKAT_AUTOTURN.Value,
E_0, E_1, E_2, E_K, E_K_1, E_K_2, C_PID, CENTER_POINT, HEADING_DIFF
    End If  'end of log data block

End If

'RECORD GPS TIME INFORMATION
```

```
      If SA = 6 Then
         GPS_HOUR = DATA_ARRAY(1)
         GPS_MINUTE = DATA_ARRAY(2)
         GPS_SECOND = DATA_ARRAY(3)
         GPS_TIME = GPS_HOUR & ":" & GPS_MINUTE & ":" & GPS_SECOND
         TIME_TEXT.Text = GPS_HOUR & ":" & GPS_MINUTE & ":" & GPS_SECOND
         SA = 0
      End If

      'RECORD GPS SATELLITE STATISTICS
      If SA = 7 Then
         GPS_QUALITY = DATA_ARRAY(1)
         NUMSAT = DATA_ARRAY(2)
         GPS_QUALITY_TEXT.Text = GPS_QUALITY
         NUMSAT_TEXT.Text = NUMSAT
         SA = 0
      End If
   Wend
End Sub
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub RDUP_Click()
If RDUP.Value = 1 Then
   UKAT_AUTOSTEER.Value = 1
   UKAT_AUTOTURN.Value = 1
   ROUNDUP_HEADING = False
End If
If RDUP.Value = 0 Then
   UKAT_AUTOSTEER.Value = 0
   UKAT_AUTOTURN.Value = 0
End If
End Sub
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub RDUP_VALUES_Click()
'THIS ROUTINE ALLOWS FOR APPLYING PREDEFINED A AND B POINTS WHEN USED FOR
'DEMONSTRATIONAL PURPOSES

X1 = 718490.411438581      'ROUNDUP
Y1 = 4211421.57805987

POINT_A_EAST_TEXT.Text = X1
POINT_A_NORTH_TEXT.Text = Y1

SET_B_BUTTON.Enabled = True
CANCEL_BUTTON.Enabled = True
SET_A_BUTTON.Enabled = False

X2 = 718466.491010205      'ROUNDUP
Y2 = 4211364.53880073

POINT_B_EAST_TEXT.Text = X2
POINT_B_NORTH_TEXT.Text = Y2

ACCEPT_BUTTON.Enabled = True
SET_B_BUTTON.Enabled = False
End Sub
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub SET_A_BUTTON_Click()
'SET_A_BUTTON STORES CURRENT POSITION AS A POINT IN GUIDANCE PATH

X1 = ESTNG
Y1 = NRTNG
POINT_A_EAST_TEXT.Text = X1
POINT_A_NORTH_TEXT.Text = Y1
SET_B_BUTTON.Enabled = True
CANCEL_BUTTON.Enabled = True
SET_A_BUTTON.Enabled = False

End Sub
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------
Private Sub SET_B_BUTTON_Click()
```

```
'SET_B_BUTTON STORES CURRENT POSITION AS B POINT IN GUIDANCE PATH

X2 = ESTNG
Y2 = NRTNG
POINT_B_EAST_TEXT.Text = X2
POINT_B_NORTH_TEXT.Text = Y2
ACCEPT_BUTTON.Enabled = True
SET_B_BUTTON.Enabled = False

End Sub
```



Appendix C: Node Program Code

Three Point Hitch Node

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;	AUTHOR:  MATTHEW J. DARR
;
;	TITLE:  UK_3PT_F.BAS
;
;	DATE CREATED:  6/13/2003
;
;	DESCRIPTION:  THIS PROGRAM CONTROLS THE 3PT HITCH NODE
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


DEFINE OSC 20			;DEFINE 20 MHZ OSCILLATOR

TRISB = %00001000		;SET CAN DATA DIRECTION REGISTER

;CONFIGURE ANALOG TO DIGITAL CONVERTER
TRISA = 255
ADCON1 = 2
DEFINE ADC_BITS 8

;CONFIGURE CAN BAUD RATE
CANCON = %10000000
BRGCON1 = %00000011
BRGCON2 = %00010000
BRGCON3 = %00000000

;CONFIGURE RECEIVE FILTERS AND MASKS
CIOCON = %00010000
RXB0CON = %00000000
RXF0SIDH = %10101010
RXF0SIDL = %10101010
RXF0EIDH = %10101010
RXF0EIDL = %00000011
RXM0SIDH = 0
RXM0SIDL = 0
RXM0EIDH = 0
RXM0EIDL = %11111111

CANCON = 0

;ASSIGN PROGRAM VARIABLES
DEAD_BAND VAR BYTE
HST_SPEED VAR BYTE
DEAD_POS VAR WORD
DEAD_NEG VAR WORD
DIR VAR PORTC.1
PWM_PIN VAR PORTC.2
BRAKE VAR PORTC.3
SPEED_MSG VAR RXB0CON.0
ON_TIME VAR WORD
CURRENT_SPEED VAR BYTE

;SET INITIAL PROGRAM VALUES
DEAD_BAND = 5
HST_SPEED = 112
DEAD_POS = (HST_SPEED + DEAD_BAND)
DEAD_NEG = (HST_SPEED - DEAD_BAND)
PAUSE 1000

HIGH BRAKE

LOOP:
```

```
IF (PIR3.0 = 1) AND (SPEED_MSG = 0) THEN      ;RECEIVE NEW HITCH MESSAGE
        HST_SPEED = (RXB0D0*1)
        RXB0CON = 0
        RXB0EIDL = 0
        DEAD_POS = (HST_SPEED + DEAD_BAND)
        DEAD_NEG = (HST_SPEED - DEAD_BAND)
ENDIF

ADCIN 0, CURRENT_SPEED                                  ;CALCULATE CURRENT SPEED

;DIAGNOSTICS
SEROUT2 PORTB.7, 16468, [254,1,254,128, DEC HST_SPEED, 44, DEC CURRENT_SPEED]

IF (CURRENT_SPEED <= DEAD_POS) AND (CURRENT_SPEED >= DEAD_NEG) THEN ;DON'T ADJUST
        HIGH BRAKE
        HIGH DIR
        HIGH PWM_PIN
        BRANCHL 0,[LOOP]
ENDIF

IF (DEAD_NEG < CURRENT_SPEED) THEN       ;LOWER HITCH
        HIGH PWM_PIN
        LOW DIR
        LOW BRAKE
        BRANCHL 0,[LOOP]
ENDIF

IF (DEAD_POS > CURRENT_SPEED) THEN       ;RAISE HITCH
        HIGH PWM_PIN
        HIGH DIR
        LOW BRAKE
        BRANCHL 0,[LOOP]
ENDIF

BRANCHL 0,[LOOP]
BRANCHL 0,[LOOP]
GOTO LOOP
END
```

Computer Receive Node

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;		AUTHOR:  MATTHEW J. DARR
;
;		TITLE:  UK_CPU_F.BAS
;
;		DATE CREATED:  6/13/2003
;
;		DESCRIPTION:  THIS PROGRAM RECEIVES MESSAGES FROM THE CAN BUS AND RETRANSMITS
;		THE MESSAGES VIA RS232
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DEFINE OSC 20				;DEFINE 20 MHZ OSCILLATOR

;DEFINE VARIABLES USED IN THE PROGRAM
HOUR VAR BYTE
MINUTE VAR BYTE
SECOND VAR BYTE
LATDD VAR WORD
LATMM VAR WORD
LATMMM1 VAR WORD
LATMMM2 VAR WORD
LATDIR VAR WORD
LONDD VAR WORD
LONMM VAR WORD
LONMMM1 VAR WORD
LONMMM2 VAR WORD
LONDIR VAR WORD
GPS_QUALITY VAR BYTE
NUMSAT VAR BYTE
LONDD_OFFSET VAR WORD
LONDD_SCALED VAR WORD
SA VAR BYTE
TIMEOUT VAR BYTE
LON_WORD1 VAR WORD
LON_WORD2 VAR WORD
LAT_WORD1 VAR WORD
LAT_WORD2 VAR WORD
TX0 VAR LAT_WORD1.BYTE0
TX1 VAR LAT_WORD1.BYTE1
TX2 VAR LAT_WORD2.BYTE0
TX3 VAR LAT_WORD2.BYTE1
TX4 VAR LON_WORD1.BYTE0
TX5 VAR LON_WORD1.BYTE1
TX6 VAR LON_WORD2.BYTE0
TX7 VAR LON_WORD2.BYTE1
TURN_RATE VAR WORD
TR0 VAR TURN_RATE.BYTE0
TR1 VAR TURN_RATE.BYTE1
TURN_STATUS VAR BYTE

INTCON = %00000000			;TURN OFF INTERRUPTS
PIE3 = %11111111
TRISB = %00001000			;ENABLE THE CAN RECEIVE PIN FOR INPUT AND CAN TRANSMIT
					PIN FOR OUTPUT

;CONFIGURE THE CAN BAUD RATE
CANCON = %10000000
BRGCON1 = %00000011
BRGCON2 = %00010000
BRGCON3 = %00000000

;CONFIGURE THE CAN RECEIVE DATA REGISTERS
CIOCON = %00010000
RXB0CON = %00000000
RXF0SIDH = %10101010
```

```
RXF0SIDL = %10101010
RXF0EIDH = %10101010
RXF0EIDL = %00000001
RXM0SIDH = %00000000
RXM0SIDL = %00000000
RXM0EIDH = %00000000
RXM0EIDL = %10000000                    ;BIT 7 OF EIDL MUST BE ZERO FOR MESSAGE ACCEPTANCE

CANCON = %00000000


LOOP2:  IF PIR3.0 = 1 THEN              ;CHECK FOR NEW CAN MESSAGE
                SA = RXB0EIDL           ;STORE NEW MESSAGE IN GIVEN VARIABLES
                TX0 = RXB0D0
                TX1 = RXB0D1
                TX2 = RXB0D2
                TX3 = RXB0D3
                TX4 = RXB0D4
                TX5 = RXB0D5
                TX6 = RXB0D6
                TX7 = RXB0D7
                RXB0CON = %00000000     ;RESET RECEIVER BUFFER
                PIR3 = 0                ;CLEAR INTERRUPT BIT
        ENDIF

        TRISB = %00001000
                SEROUT2 PORTB.7, 16468, [254, 1, 254, 128, DEC TRISB]

        IF SA = 5 THEN                                  ;LAT AND LON INFORMATION
                LATDD = LAT_WORD1 / 1000                ;CONDITION GPS INFORMATION FOR RS232
TRANSMISSION
                LATMMM1 = LAT_WORD1 - (LATDD * 1000)
                LATMM = LAT_WORD2 / 1000
                LATMMM2 = LAT_WORD2 - (LATMM * 1000)
                LONDD_SCALED = LON_WORD1 / 1000
                LONDD = LONDD_SCALED + 40
                LONMMM1 = LON_WORD1 - (LONDD_SCALED * 1000)
                LONMM = LON_WORD2 / 1000
                LONMMM2 = LON_WORD2 - (LONMM * 1000)    ;OUTPUT MESSAGE VIA RS232
                SEROUT2 PORTC.7, 16416, ["$", DEC SA, 44, DEC LATDD, 44, DEC LATMM, 44, DEC LATMMM1, 44, DEC
                        LATMMM2, 44, DEC LONDD, 44, DEC LONMM, 44, DEC LONMMM1, 44, DEC LONMMM2, "#", 10,
                        13]
                SA = 0                                  ;RESET THE SOURCE ADDRESS VARIABLE
        ENDIF

        IF SA = 6 THEN                          ;TIME INFORMATION
                HOUR = TX0
                MINUTE = TX1
                SECOND = TX2
                SEROUT2 PORTC.7, 16416, ["$", DEC SA, 44, DEC HOUR, 44, DEC MINUTE, 44, DEC SECOND, "#", 10, 13]
                SA = 0
                SEROUT2 PORTB.7, 16468, [254, 1, 254, 128, DEC MINUTE, 44, DEC SECOND, 44, DEC HOUR]
        ENDIF

        IF SA = 7 THEN                          ;GPS QUALITY INFORMATION
                GPS_QUALITY = TX0
                NUMSAT = TX1
                SEROUT2 PORTC.7, 16416, ["$", DEC SA, 44, DEC GPS_QUALITY, 44, DEC NUMSAT, "#", 10, 13]
                SA = 0
        ENDIF

        IF SA = 9 THEN                          ;HEADLAND TURN STATUS
                TURN_STATUS = TX0
                SEROUT2 PORTC.7, 16416, ["$", DEC SA, 44, DEC TURN_STATUS, "#",10,13]
                SA = 0
        ENDIF

        IF (SA = %00010010) THEN                ;GYRO SENSOR OUTPUT MESSAGE
                TR0 = TX0
                TR1 = TX1
```

```
            SEROUT2 PORTC.7, 16416, ["$", DEC SA, 44, DEC TURN_RATE, "#", 10,13]
            SA = 0
        ENDIF

        BRANCHL 0, [LOOP2]                          ;RETURN TO BEGINNING OF THE LOOP
        GOTO LOOP2

END
```

Computer Transmitter Node

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;           AUTHOR:             MATTHEW DARR
;           TITLE:      UK_GPS.BAS
;           PURPOSE:  READ A RS232 OUTPUT FROM A LAPTOP AND ASSERT THE MESSAGE ONTO THE CAN
;           CREATED:  6/13/2003
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;           FORMAT:  THE RS232 MESSAGE IS FORMATTED AS $SA,DATA1,DATA2,DATA3#
;                    WHERE DATA IS A WORD VARIABLE
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


DEFINE OSC 20                           ;DEFINE 20 MHZ OSCILLATOR

;DEFINE VARIABLES
SA VAR BYTE
DATA1 VAR WORD
H0 VAR BYTE
H1 VAR BYTE
I VAR BYTE
DATA2 VAR BYTE
DATA3 VAR BYTE

TRISB = %00001000                       ;CONFIGURE CAN PORT DATA DIRECTION REGISTER

;CONFIGURE CAN BAUD RATE
CANCON = %10001000
BRGCON1 = %00000011
BRGCON2 = %00010000
BRGCON3 = %00000000
CIOCON = %00010000

CANCON = %00000110

;DEFINE UPPER THREE BYTES OF CAN MESSAGE
TXB1SIDH = %10101010
TXB1SIDL = %10101010
TXB1EIDH = %10101010


LOOP:   SERIN2 PORTC.0, 16416, [WAIT("$"), DEC SA, DEC DATA1, DEC DATA2, DEC DATA3]
                                        ;WAIT FOR NEW MESSAGE FROM TASK COMPUTER
        TXB1EIDL = SA                   ;ASSIGN SOURCE ADDRESS
        H0 = DATA1.BYTE0
        H1 = DATA1.BYTE1
        TXB1D0 = H0                     ;PLACE VARIABLES INTO CAN TRANSMIT REGISTER
        TXB1D1 = H1
        TXB1D2 = DATA2
        TXB1D3 = DATA3
        TXB1DLC = %00000100             ;ASSIGN DATA LENGTH CODE OF 8 BYTES
        TXB1CON = %00001011             ;SEND MESSAGE WITH HIGHEST PRIORITY

        WHILE TXB1CON.3 = 1             ;WAIT FOR CAN MESSAGE TO TRANSMIT BEFORE RETURNING
                I = 1                   ;GIVES THE LOOP SOMETHING TO DO
        WEND

        BRANCHL 0, [LOOP]               ;BRANCH TO BEGINNING OF THE LOOP
        GOTO LOOP

END
```

Global Positioning System Node

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;		AUTHOR:		MATTHEW DARR
;		TITLE:	UK_GPS_F.BAS
;		PURPOSE:  DECODE A GPS SIGNAL AND TRANSMIT LATITUDE, LONGITUDE, TIME, GPS QUALITY,
;		AND NUMBER OF SATELLITES DATA
;		CREATED:  6/13/2003
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DEFINE OSC 20					;DEFINE 20 MHZ OSCILLATOR

;DEFINE VARIABLES
I VAR BYTE
HOUR VAR BYTE
MINUTE VAR BYTE
SECOND VAR BYTE
LATDD VAR WORD
LATMM VAR WORD
LATMMM1 VAR WORD
LATMMM2 VAR WORD
LATDIR VAR WORD
LONDD VAR WORD
LONMM VAR WORD
LONMMM1 VAR WORD
LONMMM2 VAR WORD
LONDIR VAR WORD
GPS_QUALITY VAR BYTE
NUMSAT VAR BYTE
LONDD_OFFSET VAR WORD
LONDD_SCALED VAR WORD
LON_WORD1 VAR WORD
LON_WORD2 VAR WORD
LAT_WORD1 VAR WORD
LAT_WORD2 VAR WORD
LAT_TX1 VAR LAT_WORD1.BYTE0
LAT_TX2 VAR LAT_WORD1.BYTE1
LAT_TX3 VAR LAT_WORD2.BYTE0
LAT_TX4 VAR LAT_WORD2.BYTE1
LON_TX1 VAR LON_WORD1.BYTE0
LON_TX2 VAR LON_WORD1.BYTE1
LON_TX3 VAR LON_WORD2.BYTE0
LON_TX4 VAR LON_WORD2.BYTE1

TRISB = %00001000				;CONFIGURE CAN PORT DATA DIRECTION REGISTER

;CONFIGURE CAN BAUD RATE
CANCON = %10001000
BRGCON1 = %00000011
BRGCON2 = %00010000
BRGCON3 = %00000000
CIOCON = %00010000
PIE3 = %11111111

CANCON = %00000110

;DEFINE UPPER THREE BYTES OF CAN MESSAGE
TXB1SIDH = %10101010
TXB1SIDL = %10101010
TXB1EIDH = %10101010


LOOP:	SERIN2 PORTC.0, 16416, 1000, LOOP, [WAIT("$GPGGA,"), DEC2 HOUR, DEC2 MINUTE, DEC2 SECOND, WAIT(","),
		DEC2 LATDD, DEC2 LATMM, WAIT("."), DEC3 LATMMM1, DEC3 LATMMM2, WAIT(","), LATDIR, WAIT(","),
		DEC3 LONDD, DEC2 LONMM, WAIT("."), DEC3 LONMMM1, DEC3 LONMMM2, WAIT(","), LONDIR,
		WAIT(","), DEC1 GPS_QUALITY, WAIT(","), DEC2 NUMSAT]
	LAT_WORD1 = (LATDD * 1000) + LATMMM1		;FORMATS LAT VALUES INTO 4 BYTES FOR CAN
							TRANSMISSION
	LAT_WORD2 = (LATMM * 1000) + LATMMM2
```

```
        LONDD_OFFSET = 40                   ;OFFSETS LON DEGREE BY 40 TO FIT INTO A WORD VARIABLE
        LONDD_SCALED = LONDD - LONDD_OFFSET
        LON_WORD1 = (LONDD_SCALED * 1000) + LONMMM1          ;FORMATS LON VALUES INTO 4 BYTES FOR
                                                              CAN TRANSMISSION
        LON_WORD2 = (LONMM * 1000) + LONMMM2
        TXB1EIDL = %00000101                ;ASSIGN SOURCE ADDRESS = 5
        TXB1D0 = LAT_TX1                    ;PLACE POSITION VALUES INTO CAN TRANSMIT REGISTER
        TXB1D1 = LAT_TX2
        TXB1D2 = LAT_TX3
        TXB1D3 = LAT_TX4
        TXB1D4 = LON_TX1
        TXB1D5 = LON_TX2
        TXB1D6 = LON_TX3
        TXB1D7 = LON_TX4
        TXB1DLC = %00001000                 ;ASSIGN DATA LENGTH CODE OF 8 BYTES
        TXB1CON = %00001011                 ;SEND MESSAGE WITH HIGHEST PRIORITY


        WHILE TXB1CON.3 = 1                 ;WAIT FOR CAN MESSAGE TO TRANSMIT BEFORE RETURNING
                I = 1                       ;GIVES THE LOOP SOMETHING TO DO
        WEND

        PAUSE 30
        TXB1EIDL = %00000110                ;ASSIGN SOURCE ADDRESS = 6
        TXB1D0 = HOUR
        TXB1D1 = MINUTE
        TXB1D2 = SECOND
        TXB1DLC = %00000011
        TXB1CON = %00001011                 ;SEND TIME INFORMATION

        WHILE TXB1CON.3 = 1         ;WAIT FOR CAN MESSAGE TO TRANSMIT BEFORE RETURNING
                I = 1
        WEND

        PAUSE 30
        TXB1EIDL = %00000111                ;ASSIGN SOURCE ADDRESS = 7
        TXB1D0 = GPS_QUALITY
        TXB1D1 = NUMSAT
        TXB1DLC = %00000010
        TXB1CON = %00001011                 ;SEND SATELLITE INFORMATION

        WHILE TXB1CON.3 = 1                 ;WAIT FOR CAN MESSAGE TO TRANSMIT BEFORE RETURNING
                I = 1
        WEND

        BRANCHL 0, [LOOP]                   ;RETURN TO BEGINNING OF THE LOOP
        GOTO LOOP

END
```

Hydrostatic Transmission Control Node

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;		AUTHOR:  MATTHEW J. DARR
;
;		TITLE:  UK_HT_F.BAS
;
;		DATE CREATED:  6/13/2003
;
;		DESCRIPTION:  THIS PROGRAM ACTS AS THE CONTROL NODE FOR THE HYDROSTATIC TRANS
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DEFINE OSC 20				;DEFINE 20 MHZ OSCILLATOR

TRISB = %00001000			;DEFINE CAN DATA DIRECTION REGISTERS

TRISA = 255				;CONFIGURE ANALOG TO DIGITAL CONVERTER
ADCON1 = 2
DEFINE ADC_BITS 8

;CONFIGURE CAN BAUD RATE
CANCON = %10000000
BRGCON1 = %00000011
BRGCON2 = %00010000
BRGCON3 = %00000000

;CONFIGURE RECEIVE BUFFER FILTERS AND MASK
CIOCON = %00010000
RXB0CON = %00000000
RXF0SIDH = %10101010
RXF0SIDL = %10101010
RXF0EIDH = %10101010
RXF0EIDL = %00000010
RXM0SIDH = 0
RXM0SIDL = 0
RXM0EIDH = 0
RXM0EIDL = %11111111

CANCON = 0

;SET INITIAL VARIABLES
DEAD_BAND VAR BYTE
HST_SPEED VAR BYTE
DEAD_POS VAR WORD
DEAD_NEG VAR WORD
DIR VAR PORTC.1
PWM_PIN VAR PORTC.2
BRAKE VAR PORTC.3
SPEED_MSG VAR RXB0CON.0
ON_TIME VAR WORD
CURRENT_SPEED VAR BYTE

;SET INTITIAL PARAMETERS
DEAD_BAND = 5
HST_SPEED = 112
DEAD_POS = (HST_SPEED + DEAD_BAND)
DEAD_NEG = (HST_SPEED - DEAD_BAND)
PAUSE 1000

HIGH BRAKE

LOOP:
IF (PIR3.0 = 1) AND (SPEED_MSG = 0) THEN	;RECEIVE NEW SPEED MESSAGE
		HST_SPEED = (RXB0D0*1)
		RXB0CON = 0
		RXB0EIDL = 0
		DEAD_POS = (HST_SPEED + DEAD_BAND)
```

```
        DEAD_NEG = (HST_SPEED - DEAD_BAND)
ENDIF

IF (PIR3.0 = 1) AND (RXB0CON.0 = 1) THEN        ;ADJUST CONTROL PARAMETERS
        DEAD_BAND = RXB0D0
        ON_TIME = (RXB0D2)*1000
        RXB0CON = 0
        DEAD_POS = (HST_SPEED + DEAD_BAND)
        DEAD_NEG = (HST_SPEED - DEAD_BAND)
        GOTO LOOP
ENDIF

ADCIN 0, CURRENT_SPEED              ;CALCULATE CURRENT SPEED

SEROUT2 PORTB.7, 16468, [254,1,254,128, DEC HST_SPEED, 44, DEC CURRENT_SPEED]        ;DIAGNOSTICS

IF (CURRENT_SPEED <= DEAD_POS) AND (CURRENT_SPEED >= DEAD_NEG) THEN ;DON'T ADJUST
        HIGH BRAKE
        HIGH DIR
        HIGH PWM_PIN
        BRANCHL 0,[LOOP]
ENDIF

IF (DEAD_NEG > CURRENT_SPEED) THEN                  ;SLOW DOWN
        HIGH PWM_PIN
        HIGH DIR
        LOW BRAKE
        BRANCHL 0,[LOOP]
ENDIF

IF (DEAD_POS < CURRENT_SPEED) THEN                  ;SPEED UP
        HIGH PWM_PIN
        LOW DIR
        LOW BRAKE
        BRANCHL 0,[LOOP]
ENDIF

BRANCHL 0,[LOOP]
BRANCHL 0,[LOOP]
GOTO LOOP
END
```

Radio Controller Node

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;		AUTHOR:  MATTHEW J. DARR
;
;		TITLE:  UK_RF_F.BAS
;
;		DATE CREATED:  6/13/2003
;
;		DESCRIPTION:  THIS PROGRAM ACTS AS THE INTERFACE BETWEEN THE RADIO CONTROLLER AND
;		THE CAN BUS
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DEFINE OSC 20				;DEFINE 20 MHZ OSCILLATOR

;DEFINE VARIABLES
WIDTH VAR WORD
WIDTH2 VAR WORD
LENGTH VAR WORD
STEERING_ANGLE VAR WORD
STEER_FB VAR WORD
HYDRO_SPEED VAR WORD
HYDRO_FB VAR WORD
H0 VAR BYTE
H1 VAR BYTE
H2 VAR BYTE
H3 VAR BYTE
ON_OFF VAR WORD
HYDRO_FB_8 VAR BYTE
I VAR BYTE

TRISB = %00001000			;SET CAN DATA DIRECTION PINS

;ENABLE AND CONFIGURE THE ANALOG TO DIGITAL CONVERTER
TRISA = 255				;CONFIGURE PORTA FOR INPUTS
ADCON1 = 128				;RIGHT JUSTIFY THE RESULT REGISTER
DEFINE ADC_BITS 10			;SET ADCONV AT 10 BIT

;CONFIGURE THE CAN BAUD RATE
CANCON = %10001000
BRGCON1 = %00000011
BRGCON2 = %00010000
BRGCON3 = %00000000
CIOCON = %00010000
PIE3 = %11111111

CANCON = %00000110

LOOP:	PULSIN PORTC.5, 1, ON_OFF		;READ FIRST RADIO CHANNEL - ON/OFF OPERATION SWITCH
	IF ON_OFF < 750 THEN		;NOT OPERATIONAL
		PAUSE 100
		GOTO LOOP
	ENDIF

	PULSIN PORTC.7, 1, STEERING_ANGLE		;READ STEERING CHANNEL
	STEER_FB = (43*STEERING_ANGLE/100 - 253)*6
	IF STEER_FB > 900 THEN			;IF OVER MAX THEN STEER STRAIGHT
		STEER_FB = 470
	ENDIF
	IF STEER_FB < 100 THEN			;IF UNDER MIN THEN STEER STRAIGHT
		STEER_FB = 470
	ENDIF
	H0 = STEER_FB.BYTE0
	H1 = STEER_FB.BYTE1
	TXB1D0 = H0				;TRANSMIT STEERING MESSAGE
	TXB1D1 = H1
	TXB1DLC = %00000010
```

```
        TXB1SIDH = %10101010
        TXB1SIDL = %10101010
        TXB1EIDH = %10101010
        TXB1EIDL = %00000001
        TXB1CON = %00001011

        PAUSE 20

        PULSIN PORTC.6, 1, HYDRO_SPEED          ;READ SPEED CHANNEL
        HYDRO_FB = (61*HYDRO_SPEED/100 - 337)*4
        IF HYDRO_FB > 960 THEN                  ;IF OVER MAX THEN STOP VEHICLE
                HYDRO_FB = 450
        ENDIF
        IF HYDRO_FB < 100 THEN                  ;IF UNDER MIN THEN STOP VEHICLE
                HYDRO_FB = 450
        ENDIF
        HYDRO_FB_8 = (HYDRO_FB / 4)             ;CONVERT DATA TO 8 BIT
        H2 = HYDRO_FB.BYTE0
        H3 = HYDRO_FB.BYTE1
        TXB1D0 = HYDRO_FB_8                     ;TRANSMIT SPEED MESSAGE
        TXB1DLC = %00000001
        TXB1SIDH = %10101010
        TXB1SIDL = %10101010
        TXB1EIDH = %10101010
        TXB1EIDL = %00000010
        TXB1CON = %00001011

        WHILE TXB1CON.3 = 1             ;WAIT FOR CAN MESSAGE TO TRANSMIT BEFORE RETURNING
                I = 1                   ;GIVES THE LOOP SOMETHING TO DO
        WEND

        PAUSE 20
        SEROUT2 PORTB.7, 16468, [254, 1, 254, 128, DEC STEERING_ANGLE, 44, DEC HYDRO_FB_8]   ;DIAGNOSTICS
        BRANCHL 0,[LOOP]                        ;RETURN TO BEGINNING OF LOOP
        GOTO LOOP

END
```

Steering Node

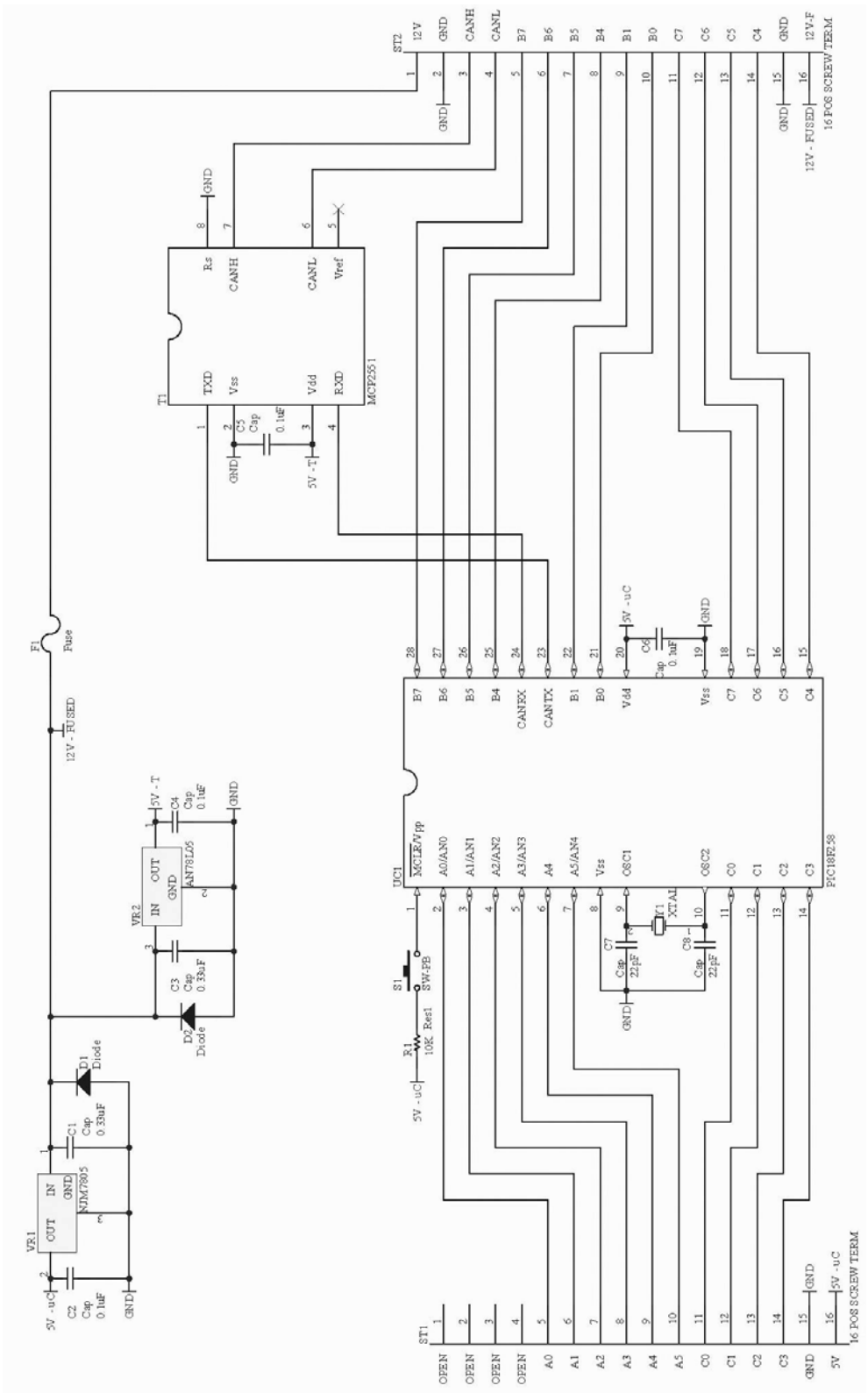
```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;           AUTHOR:  MATTHEW J. DARR
;
;           TITLE:  UK_ST_F.BAS
;
;           DATE CREATED:  6/13/2003
;
;           DESCRIPTION:  THIS PROGRAM OPERATES THE STEERING NODE
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

DEFINE OSC 20                              ;DEFINE 20 MHZ OSCILLATOR

;DEFINE VARIABLES
SA VAR BYTE
STEERING_ANGLE VAR WORD
H0 VAR STEERING_ANGLE.BYTE0
H1 VAR STEERING_ANGLE.BYTE1
DEAD_POS VAR WORD
DEAD_NEG VAR WORD
DEAD_BAND VAR WORD
CURRENT_ANGLE VAR WORD
ON_TIME VAR WORD
DISTANCE VAR WORD
SDO VAR PORTC.3
SCLK VAR PORTC.4
PC VAR PORTC.5
SS VAR PORTC.6
HEADING VAR WORD
RS232_SEND VAR PORTC.0
RESET VAR PORTC.7
ZEROS VAR WORD
EXCESS VAR BYTE
TURN_DIR VAR BYTE
EXIT_STEER_LOOP VAR BYTE
LOOP_DIR VAR BYTE
RETURN_VALUE VAR BYTE
PHI VAR WORD
THETA1 VAR WORD
THETA2 VAR WORD
THETA3 VAR WORD
COMPASS_RETURN VAR BYTE
INT_HEAD VAR WORD
FILTER_HIT VAR BYTE
ANGLE_LOW VAR BYTE
ANGLE_HIGH VAR BYTE
AHI VAR PORTB.0
BHI VAR PORTB.1
ALI VAR PORTB.5
BLI VAR PORTB.6
DISABLE_BRAKE VAR PORTB.4

TRISB = %00001000                          ;CONFIGURE THE CAN DATA DIRECTION REGISTER
TRISC = %00000000                          ;CONFIGURE PORTC AS OUTPUTS

;ENABLE AND CONFIGURE THE ANALOG TO DIGITAL CONVERTER
TRISA = 255                                ;CONFIGURE PORTA AS INPUTS
ADCON1 = 128                               ;RIGHT JUSTIFIES RESULTS REGISTER
DEFINE ADC_BITS 10                         ;SET ADCONV TO 10 BIT MODE

;CONFIGURE THE CAN BAUD RATE
CANCON = %10000000
BRGCON1 = %00000011
BRGCON2 = %00010000
BRGCON3 = %00000000
```

```
;SET UP CAN RECEIVE BUFFER 0 AND FILTER 0 AND MASK 1
CIOCON = %00010000
RXB0CON = %00000000
RXF0SIDH = %10101010
RXF0SIDL = %10101010
RXF0EIDH = %10101010
RXF0EIDL = %00010001
RXM0SIDH = %00000000
RXM0SIDL = %00000000
RXM0EIDH = %00000000
RXM0EIDL = %11111111

RXF1SIDL = %10101010
RXF1SIDH = %10101010
RXF1EIDH = %10101010
RXF1EIDL = %00000001
RXM1SIDH = %00000000
RXM1SIDL = %00000000
RXM1EIDH = %00000000
RXM1EIDL = %11111111

;SET UP CAN RECEIVE BUFFER 1, FILTER 2, AND MASK 1
RXB1CON = %00000000
RXF2SIDH = %10101010
RXF2SIDL = %10101010
RXF2EIDH = %10101010
RXF2EIDL = %00001000

RXF3SIDH = %10101010
RXF3SIDL = %10101010
RXF3EIDH = %10101010
RXF3EIDL = %00001010

RXM1SIDH = %00000000
RXM1SIDL = %00000000
RXM1EIDH = %00000000
RXM1EIDL = %11111111

;CONFIGURE THE HIGH THREE BYTES OF THE CAN TRANSMIT INDENTIFIER
TXB1SIDH = %10101010
TXB1SIDL = %10101010
TXB1EIDH = %10101010

CANCON = %00000000                    ;SET CONTROL REGISTER TO NORMAL OPERATION MODE

;SET INITIAL PROGRAM VARIABLES
DEAD_BAND = 5
STEERING_ANGLE = 470
DEAD_POS = (STEERING_ANGLE + DEAD_BAND)
DEAD_NEG = (STEERING_ANGLE - DEAD_BAND)
PAUSE 1000

;INITIALIZE COMPASS
HIGH SS
HIGH PC
HIGH RESET

;RESET COMPASS
HIGH PC
HIGH SS
PAUSE 5
LOW RESET
PAUSE 15
HIGH RESET
PAUSE 500

;RESET LOOP DIRECTION PARAMETERS
RETURN_VALUE = 0
LOOP_DIR = 0
EXIT_STEER_LOOP = 0
```

```
;SET INITIAL PROGRAM VARIABLES
ANGLE_LOW = 165
ANGLE_HIGH = 190
HIGH DISABLE_BRAKE

LOOP:   IF (RXB0CON.7 = 1) AND (RXB0CON.0 = 1) THEN          ;CHECK FOR NEW STEERING MESSAGE IN BUFFER ZERO
                H0 = RXB0D0                                  ;STORE DATA INTO VARIABLES
                H1 = RXB0D1
                PIR3.0 = 0                                   ;RESET INTERRUPT FLAGS
                RXB0CON = 0                                  ;RESET NEW MESSAGE RECEIVED FLAG
                DEAD_POS = (STEERING_ANGLE + DEAD_BAND)      ;CALCULATE DEADBAND LIMITS
                DEAD_NEG = (STEERING_ANGLE - DEAD_BAND)
        ENDIF

        IF PIR3.1 = 1 THEN                                   ;CHECK FOR NEW CONFIG MESSAGE IN BUFFER ONE
                FILTER_HIT = RXB1CON & %00000111
                IF FILTER_HIT = 2 THEN                       ;CHECK FOR HEADLAND TURN MESSAGE
                        TURN_DIR = RXB1D2                    ;STORE DATA INTO VARIABLES
                        H0 = RXB1D0
                        H1 = RXB1D1
                        PIR3.1 = 0
                        RXB1CON = 0
                        DEAD_POS = (STEERING_ANGLE + DEAD_BAND)
                        DEAD_NEG = (STEERING_ANGLE - DEAD_BAND)
                        BRANCHL TURN_DIR,[LOOP,TURN]
                ENDIF
                IF FILTER_HIT = 3 THEN                       ;CHECK FOR NEW CONFIG MESSAGE
                        DEAD_BAND = RXB1D0
                        ANGLE_LOW = RXB1D2
                        ANGLE_HIGH = RXB1D3
                        PIR3.1 = 0
                        RXB1CON = 0
                        BRANCHL 0,[LOOP]
                ENDIF
        ENDIF

        ADCIN 0, CURRENT_ANGLE                               ;CAPTURE CURRENT STEERING ANGLE

        SEROUT2 PORTB.7, 16468, [254, 1, 254, 128, DEC STEERING_ANGLE, 44, DEC CURRENT_ANGLE]
        ;DIAGNOSTIC DATA

        IF (CURRENT_ANGLE <= DEAD_POS) AND (CURRENT_ANGLE >= DEAD_NEG) THEN ;DON'T STEER
                HIGH ALI
                HIGH BLI
                LOW DISABLE_BRAKE
                BRANCHL 0,[LOOP]
        ENDIF

        IF DEAD_POS < CURRENT_ANGLE THEN                     ;TURN LEFT
                HIGH ALI
                LOW BLI
                LOW DISABLE_BRAKE
                BRANCHL 0,[LOOP]
        ENDIF

        IF DEAD_NEG > CURRENT_ANGLE THEN                     ;TURN RIGHT
                LOW ALI
                HIGH BLI
                LOW DISABLE_BRAKE
                BRANCHL 0,[LOOP]
        ENDIF
        BRANCHL 0, [LOOP]


TURN:   ;READ COMPASS ROUTINE
        HIGH PC
        PAUSE 50
        LOW PC                                               ;POLL FOR NEW HEADING
```

```
        PAUSE 15
        HIGH PC
        PAUSE 150                                               ;WAIT FOR CALC TO COMPLETE
        LOW SS
        PAUSE 15
        SHIFTIN SDO,SCLK,6,[ZEROS\8,HEADING\8,EXCESS\3]   ;CLOCK IN COMPASS DATA
        HIGH SS
        IF ZEROS = 1 THEN
                HEADING = HEADING + 255
        ENDIF
        INT_HEAD = HEADING
        THETA1 = 360 - INT_HEAD
        DEAD_POS = (STEERING_ANGLE + DEAD_BAND)
        DEAD_NEG = (STEERING_ANGLE - DEAD_BAND)

TURN_ACT:                                                       ;DIAL IN STEERING ANGLE
        ADCIN 0, CURRENT_ANGLE

        IF DEAD_POS < CURRENT_ANGLE THEN                ;TURN LEFT
                HIGH ALI
                LOW BLI
                LOW DISABLE_BRAKE
                BRANCHL 0,[TURN_ACT]
        ENDIF

        IF DEAD_NEG > CURRENT_ANGLE THEN                ;TURN RIGHT
                LOW ALI
                HIGH BLI
                LOW DISABLE_BRAKE
                BRANCHL 0,[TURN_ACT]
        ENDIF

        IF (CURRENT_ANGLE <= DEAD_POS) AND (CURRENT_ANGLE >= DEAD_NEG) THEN ;DON'T STEER
                HIGH ALI
                HIGH BLI
                LOW DISABLE_BRAKE
                BRANCHL 0,[TURN_FEEDBACK]
        ENDIF

TURN_FEEDBACK:                                                  ;TURN UNTIL VEHICLE HAS GONE 180
DEGREES
        HIGH PC
        PAUSE 50
        LOW PC                                                  ;POLL FOR NEW HEADING
        PAUSE 15
        HIGH PC
        PAUSE 150                                               ;WAIT FOR CALC TO COMPLETE
        LOW SS
        PAUSE 15
        SHIFTIN SDO,SCLK,6,[ZEROS\8,HEADING\8,EXCESS\3]
        HIGH SS

        IF ZEROS = 1 THEN
                HEADING = HEADING + 255
        ENDIF

        THETA2 = 360 - HEADING
        THETA3 = ABS(THETA1-THETA2)
        PHI = 360 - THETA3
        IF (PHI > ANGLE_LOW) AND (PHI < ANGLE_HIGH) THEN
                PIR3.1 = 0
                RXB1CON = 0
                TXB1EIDL = 9
                TXB1D0 = 1
                TXB1DLC = %00000001
                TXB1CON = %00001011
                BRANCHL 0,[LOOP]
        ENDIF
        BRANCHL 0,[TURN_FEEDBACK]
END
```

Appendix D: Electronic Control Unit Circuit Diagram


ST2
12V
GND
CANH
CANL
B7
B6
B5
B4
B1
B0
C7
C6
C5
C4
GND
12V-F
12V - FUSED
16 POS SCREW TERM
T1
TXD
Vss
Vdd
RXD
Rs
CANH
CANL
Vref
MCP2551
GND
5V - T
Cap
0.1uF
F1
Fuse
12V - FUSED
VR2
IN
OUT
GND
LM78L05
VR1
LM7805
D1
Diode
D2
Diode
C1
C2
C3
C4
Cap
0.33uF
0.1uF
5V - uC
UC1
MCLR/Vpp
A0/AN0
A1/AN1
A2/AN2
A3/AN3
A4
A5/AN4
Vss
OSC1
OSC2
CANRX
CANTX
Vdd
C0
C1
C2
C3
PIC18F258
S1
SW-PB
R1
10K Res1
C7
C8
C6
22pF
XTAL
Y1
ST1
OPEN
A0
A1
A2
A3
A4
A5
C0
C1
C2
C3
GND
5V
16 POS SCREW TERM

References

Benson, E. R., T. S. Stombaugh, N. Noguchi, J. Will, and J. F. Reid. 1998. An evaluation of a geomagnetic direction sensor for vehicle guidance in precision agriculture applications. ASAE Paper No. 983203. St. Joseph, MI.: ASAE.

Benson, E. R., J. F. Reid, and Q. Zhang. 2001. Machine vision based steering system for agricultural combines. ASAE Paper No. 01-1159. St. Joseph, MI.: ASAE.

Bosch, R. 1991. CAN Specification, Version 2.0. Stuttgart, Germany: Robert Bosch GmbH.

Brown, D., and R. E. Lacey. 2002. A distributed control system for low pressure plant growth chambers. ASAE Paper No. 023078. St. Joseph, MI.: ASAE.

CAN-CIA. 2002. Application of Controller Area Networks. Erlangen, Germany.: CAN in Automation. Available at: www.can-cia.de. Accessed 17 February 2003.

Choi, C.H., D. C. Erbach, and R. J. Smith. 1990. Navigational tractor guidance system. *Transactions of the ASAE* 33(3):699-706.

DIN. 1993. Agricultural Tractors and Machinery; Interfaces for Signal Transfer. DIN Standard 9684. Berlin, Germany.

Doebelin, E.O. 1998. *System Dynamics: Modeling, Analysis, Simulation, Design.* New York, NY: Marcel Dekker, Inc.

Ge, J. 1987. Simulation of automatic control for tractor guidance. M.S. Thesis, Iowa State University Library, Ames, IA.

Grovum, M. A., and G. C. Zoerb. 1970. An automatic guidance system for farm tractors. *Transactions of the ASAE* 13(5):565-573,576.

Han, S., and Q. Zhang. 2001. Map-based Control Functions for Autonomous Tractors. ASAE Paper No. 01-1191. St. Joseph, MI.: ASAE.

Hofstee, J. W., and D. Goense. 1999. Simulation of a controller area network-based tractor-implement data bus according to ISO 11783. *Journal of Agricultural Engineering Research* 73:383-394

ISO. 2002. Tractors and machinery for agriculture and forestry – Serial control and communications data network – Part 2: Physical layer. ISO Standard 11783. Geneva, Switzerland: ISO.

ISO. 1998. Tractors and machinery for agriculture and forestry – Serial control and communications data network – Part 3: Data link layer. ISO Standard 11783. Geneva, Switzerland: ISO.

ISO. 2001. Tractors and machinery for agriculture and forestry – Serial control and communications data network – Part 5: Network layer. ISO Standard 11783. Geneva, Switzerland: ISO.

Julian, A. P. 1971. Design and performance of a steering control system for agricultural tractors. *J. Agric. Eng. Res.* 16(3): 324-336

Monson, R. J., and E. M. Dahlen. 1995. Mobile control system responsive to land area maps. U.S. Patent No. 5453924.

Noguchi, N., J. Reid, Q. Zhang, and J. Will. 2001. Turning function for robot tractor based on spline function. ASAE Paper No. 01-1196. St. Joseph, MI.: ASAE.

Ollis, M., and A. Stentz. 1996. First results in vision-based crop line tracking. In *Proceedings of the IEEE Robotics and Automation Conference*, 951-956 Minneapolis, MN.

Owen, G. M. 1982. A tractor handling study. *Veh. Sys. Dyn.* 11:215-240

Reid, J. F., Q. Zhang, N. Noguchi, and M. Dickson. 2000. Agricultural automatic guidance research in North America. *Computers and Electronics in Agriculture* 25:155-167

Reid, J. F., and S. W. Searcy. 1987. Vision-based guidance of an agricultural tractor. *IEEE Control Systems*. 7(12):39-43

SAE. 1998. Part7, Vehicle Application Layer. SAE Standard J1939. *SAE Handbook.* Warrendale, PA: SAE.

Stombaugh, T. S., E. R. Benson, and J. W. Hummel. 1999. Guidance control of agricultural vehicles at high field speeds. *Transactions of the ASAE* 42(2):537-544.

Stone, M., D. Giles, and K. Dieball. 1999a. Distributed network systems for control of spray droplet size and application rate for precision chemical application. ASAE Paper No. 993112. St. Joseph, MI.: ASAE.

Stone, M., K. McKee, C. Formwalt, and R. Benneweis. 1999b. ISO 11783: an electronic communications protocol for agricultural equipment. ASAE Distinguished Lecture #23, Agricultural Equipment Technology Conference. Louisville, Kentucky.

Tian, L., J. F. Reid, and J. W. Hummel. 1999. Development of a precision sprayer for site-specific weed management. *Transactions of the ASAE* 42(4):893-900.

Wei, J., N. Zhang, N. Wang, D. Oard, Q. Stoll, D. Lenhert, M. Neilson, M. Mizuno, and G. Sing. 2001. Design of an embedded weed-control system using controller area network (CAN). ASAE Paper No. 013033. St. Joseph, Mich.: ASAE.

Willrodt, F. L. 1924. Steering attachment for tractors. US Patent No. 1506706.

Wong, J. Y. 1978. *Theory of Ground Vehicles*. New York, NY: John Wiley & Sons, Inc.

Young, S. C. 1993. Electronic control system for GENESIS™ 70 series tractor. SAE Paper No. 941789. Warrendale, PA.: SAE

Vita

Matthew John Darr was born on October 11th, 1979 at the Coshocton County Memorial Hospital in Coshocton, Ohio. The author grew up on a rural farm in nearby Newcomerstown, Ohio. In May of 1998 he graduated with honors from Ridgewood High School, located in West Lafayette, Ohio. In September of 1998, the author entered The Ohio State University with a focus in Food, Agricultural, and Biological Engineering. The author was then awarded a Bachelor's of Science Degree in Food, Agricultural, and Biological Engineering in June of 2002. Upon completion of his B.S. degree, the author accepted a fellowship to work with Dr. Tim Stombaugh at the University of Kentucky and to pursue a Master's of Science Degree in Biosystems and Agricultural Engineering.

The author has been a member of the American Society of Agricultural Engineers (ASAE) since 1998, and was inducted into the Alpha Epsilon Society for Agricultural Engineers in the spring of 2001. He also serves as a member on the Automation Committee within ASAE.

Matthew John Darr